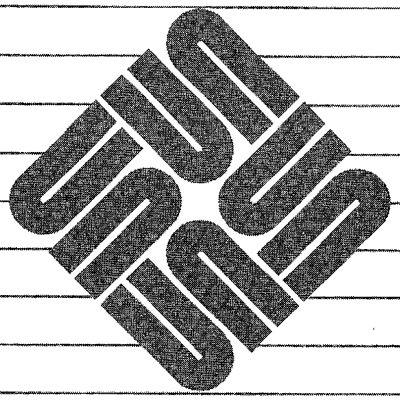
sun microsystems

# SunOS Reference Manual



Part Number: 800-3827-10
Revision A of 27 March, 1990

The Sun logo, Sun Microsystems, Sun Workstation, NFS, and TOPS are registered trademarks of Sun Microsystems, Inc.

Sun, Sun-2, Sun-3, Sun-4, Sun386*i*, SPARCstation, SPARCserver, NeWS, NSE, OpenWindows, SPARC, SunInstall, SunLink, SunNet, SunOS, SunPro, and SunView are trademarks of Sun Microsystems, Inc.

UNIX is a registered trademark of AT&T; OPEN LOOK is a trademark of AT&T.

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations, and Sun Microsystems, Inc. disclaims any responsibility for specifying which marks are owned by which companies or organizations.

This logo is a trademark of the X/Open Company Limited in the UK and other countries, and its use is licensed to Sun Microsystems, Inc. The use of this logo certifies SunOS 4.1 conformance with X/Open Portability Guide Issue 2 (XPG 2).

Copyright © 1987, 1988, 1989, 1990 Sun Microsystems, Inc. – Printed in U.S.A.

All rights reserved. No part of this work covered by copyright hereon may be reproduced in any form or by any means – graphic, electronic, or mechanical – including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

Restricted rights legend: use, duplication, or disclosure by the U.S. government is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and in similar clauses in the FAR and NASA FAR Supplement.

The Sun Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees.

This product is protected by one or more of the following U.S. patents: 4,777,485 4,688,190 4,527,232 4,745,407 4,679,014 4,435,792 4,719,569 4,550,368 in addition to foreign patents and applications pending.

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California. We acknowledge the following individuals and institutions for their role in its development: The Regents of the University of California, the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California, and Other Contributors.

# Preface

**OVERVIEW**

A man page is provided for both the naive user, and sophisticated user who is familiar with the SunOS operating system and is in need of on-line information. A man page is intended to answer concisely the question "What does it do?" The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. If a man page documents System V behavior in addition to default SunOS behavior, the System V information is presented under SYSTEM V headings. See the intro pages for more information about each section, and **man**(7) for more information about man pages in general.

**NAME**

This section gives the names of the commands or functions documented, followed by a brief description of what they do.

**CONFIG**

This section is exclusive to Section 4. It lists the configuration file lines needed to install a driver into the kernel. Where necessary, the configuration lines are presented separately for different hardware platforms.

**SYNOPSIS**

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Literal characters (commands and options) are in **bold** font and variables (arguments, parameters and substitution characters) are in *italic* font. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument *must* be specified.

... Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, '*filename* ...'.

| Separator. Only one of the arguments separated by this character can be specified at time.

**PROTOCOL**

This section occurs only in subsection 3R to indicate the protocol description file. The protocol specification pathname is always listed in **bold** font.

**AVAILABILITY**

This section briefly states any limitations on the availabilty of the command. These limitations could be hardware or software specific. Software specific ones require certain installation options.

**DESCRIPTION**

This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.

**IOCTLS**

This section appears on pages in Section 4 only. Only the device class which supplies appropriate parameters to the **ioctls**(2) system call is called **ioctls** and generates its own heading. IOCTLS for a specific device are listed alphabetically (on the man page for that specific device). IOCTLS are used for a particular class of devices all which have an **io** ending, such as **mtio**(4).

**OPTIONS**

This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

**RETURN VALUES**

If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared as **void** do not return values, so they are not discussed in RETURN VALUES.

**ERRORS**

On failure, most functions place an error code in the global variable **errno** indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

**USAGE**

This section is provided as a *guidance* on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality:

**Commands**
**Modifiers**
**Variables**
**Expressions**
**Input Grammar**

**EXAMPLES**

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as

**example%**

or if the user must be super-user,

**example#**

Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.

**ENVIRONMENT**

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

**FILES**

This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

**SEE ALSO**

This section lists references to other man pages, in-house documentation and outside publications.

**DIAGNOSTICS**

This section lists diagnostic messages with a brief explanation of the condition causing the error. Messages appear in **bold** font with the exception of variables, which are in *italic* font.

**WARNINGS**

This section lists warnings about special conditions which could seriously affect your working conditions — this is not a list of diagnostics.

**NOTES**

This section lists additional information that does not belong anywhere else on the page. It takes the form of an *aside* to the user, covering points of special interest. Critical information is never covered here.

**BUGS**

This section describes known bugs and wherever possible suggests workarounds.

**NAME**

intro – introduction to commands

**DESCRIPTION**

This section describes publicly accessible commands in alphabetic order. Commands of general utility, many with enhancements from 4.3 BSD.

Pages of special interest have been categorized as follows:

1C Commands for communicating with other systems.

1G Commands used primarily for graphics and computer-aided design.

1V System V commands. One or more of the following are true:

- The man page documents System V behavior only.
- The man page documents default SunOS behavior, and System V behavior as it differs from the default behavior. These System V differences are presented under **SYSTEM V** section headers.
- The man page documents behavior compliant with *IEEE Std 1003.1-1988* (POSIX.1).

**SEE ALSO**

- Section 8 in this manual for system administration procedures, system maintenance and operation commands, local daemons, and network-services servers.
- Section 7 in this manual for descriptions of publicly available files and macro packages for document preparation.
- Section 6 in this manual for computer games.
- *SunOS User's Guide: Getting Started*
- *SunOS User's Guide: Customizing Your Environment*
- *SunView User's Guide*
- *SunOS User's Guide: Doing More*
- *Programming Utilities and Libraries*

**DIAGNOSTICS**

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of "normal" termination) one supplied by the program, see **wait**(2V) and **exit**(2V). The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code," "exit status" or "return code," and is described only where special conventions are involved.

LIST OF COMMANDS

| Name | Appears on Page | Description |
|---|---|---|
| **acctcom** | **acctcom**(1) | search and print process accounting files |
| **adb** | **adb**(1) | general-purpose debugger |
| **addbib** | **addbib**(1) | create or extend a bibliographic database |
| **adjacentscreens** | **adjacentscreens**(1) | connect multiple screens to SunView window driver |
| **admin** | **sccs-admin**(1) | create and administer SCCS history files |
| **aedplot** | **plot**(1G) | graphics filters for various plotters |
| **alias** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **align_equals** | **textedit_filters**(1) | filters provided with **textedit**(1) |
| **apropos** | **apropos**(1) | locate commands by keyword lookup |
| **ar** | **ar**(1V) | create library archives, and add or extract files |
| **arch** | **arch**(1) | display the architecture of the current host |
| **as** | **as**(1) | assembler |
| **at** | **at**(1) | execute a command or script at a specified time |
| **atq** | **atq**(1) | display the queue of jobs to be run at specified times |
| **atrm** | **atrm**(1) | remove jobs spooled by at or batch |
| **awk** | **awk**(1) | pattern scanning and processing language |
| **banner** | **banner**(1V) | display a string in large letters |
| **bar** | **bar**(1) | create tape archives, and add or extract files |
| **basename** | **basename**(1V) | display portions of pathnames and filenames |
| **batch** | **at**(1) | execute a command or script at a specified time |
| **bc** | **bc**(1) | arbitrary-precision arithmetic language |
| **bg** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **bgplot** | **plot**(1G) | graphics filters for various plotters |
| **biff** | **biff**(1) | give notice of incoming mail messages |
| **bin-mail** | **bin-mail**(1) | an early program for processing mail messages |
| **break** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **breaksw** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **cal** | **cal**(1) | display a calendar |
| **calendar** | **calendar**(1) | a simple reminder service |
| **cancel** | **lp**(1) | send/cancel requests to a printer |
| **capitalize** | **textedit_filters**(1) | filters provided with **textedit**(1) |
| **case** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **cat** | **cat**(1V) | concatenate and display |
| **cb** | **cb**(1) | a simple C program beautifier |
| **cc** | **cc**(1V) | C compiler |
| **cd** | **cd**(1) | change working directory |
| **cdc** | **sccs-cdc**(1) | change the delta commentary of an SCCS delta |
| **cflow** | **cflow**(1V) | generate a flow graph for a C program |
| **checkeq** | **eqn**(1) | typeset mathematics |
| **checknr** | **checknr**(1) | check nroff and troff input files for errors |
| **chfn** | **passwd**(1) | change local or NIS password information |
| **chgrp** | **chgrp**(1) | change the group ownership of a file |
| **chkey** | **chkey**(1) | create or change encryption key |
| **chmod** | **chmod**(1V) | change the permissions mode of a file |
| **chsh** | **passwd**(1) | change local or NIS password information |
| **clear** | **clear**(1) | clear the terminal screen |
| **clear_colormap** | **clear_colormap**(1) | clear the colormap to make console text visible |
| **clear_functions** | **clear_functions**(1) | reset the selection service to clear stuck function keys |
| **click** | **click**(1) | enable or disable the keyboard's keystroke click |
| **clock** | **clock**(1) | display the time in an icon or window |

| | | |
|---|---|---|
| **cluster** | **cluster**(1) | find optional cluster containing a file |
| **cmdtool** | **cmdtool**(1) | run a shell (or program) using the SunView text facility |
| **cmp** | **cmp**(1) | perform a byte-by-byte comparison of two files |
| **col** | **col**(1V) | filter reverse paper motions for terminal diplay |
| **colcrt** | **colcrt**(1) | filter nroff output for a terminal lacking overstrike capability |
| **coloredit** | **coloredit**(1) | alter color map segment |
| **colrm** | **colrm**(1) | remove characters from specified columns within each line |
| **comb** | **sccs-comb**(1) | combine SCCS deltas |
| **comm** | **comm**(1) | display lines in common between two sorted lists |
| **compress** | **compress**(1) | compress or expand files, display expanded contents |
| **continue** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **cp** | **cp**(1) | copy files |
| **cpio** | **cpio**(1) | copy file archives in and out |
| **cpp** | **cpp**(1) | the C language preprocessor |
| **crontab** | **crontab**(1) | install, edit, remove or list a user's crontab file |
| **crtplot** | **plot**(1G) | graphics filters for various plotters |
| **crypt** | **crypt**(1) | encode or decode a file |
| **csh** | **csh**(1) | shell with a C-like syntax and advanced interactive features |
| **csplit** | **csplit**(1V) | split a file with respect to a given context |
| **ctags** | **ctags**(1) | create a tags file for use with ex and vi |
| **ctrace** | **ctrace**(1V) | generate a C program execution trace |
| **cu** | **cu**(1C) | connect to remote system |
| **cut** | **cut**(1V) | remove selected fields from each line of a file |
| **cxref** | **cxref**(1V) | generate a C program cross-reference |
| **date** | **date**(1V) | display or set the date |
| **dbx** | **dbx**(1) | source-level debugger |
| **dbxtool** | **dbxtool**(1) | SunView interface for dbx source-level debugger |
| **dc** | **dc**(1) | desk calculator |
| **dd** | **dd**(1) | convert and copy files with various data formats |
| **default** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **defaultsedit** | **defaultsedit**(1) | edit default settings for SunView utilities |
| **defaults_from_input** | **defaultsedit**(1) | edit default settings for SunView utilities |
| **defaults_from_input** | **input_from_defaults**(1) | update current state of the mouse and keyboard |
| **defaults_to_indentpro** | **defaultsedit**(1) | edit default settings for SunView utilities |
| **defaults_to_mailrc** | **defaultsedit**(1) | edit default settings for SunView utilities |
| **delta** | **sccs-delta**(1) | make a delta to an SCCS file |
| **deroff** | **deroff**(1) | remove nroff, troff, tbl and eqn constructs |
| **des** | **des**(1) | encrypt or decrypt data using Data Encryption Standard |
| **df** | **df**(1V) | report free disk space on file systems |
| **diff** | **diff**(1) | display line-by-line differences between pairs of text files |
| **diff3** | **diff3**(1V) | display line-by-line differences between 3 files |
| **diffmk** | **diffmk**(1) | mark differences between versions of a troff input file |
| **dircmp** | **dircmp**(1V) | compare directories |
| **dirname** | **basename**(1V) | display portions of pathnames and filenames |
| **dirs** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **dis** | **dis**(1) | object code disassembler for COFF |
| **disablenumlock** | **enablenumlock**(1) | enable or disable the numlock key |
| **domainname** | **domainname**(1) | set or display name of the current NIS domain |
| **dos** | **dos**(1) | SunView window for IBM PC/AT applications |
| **dos2unix** | **dos2unix**(1) | convert text file from DOS format to ISO format |
| **du** | **du**(1V) | display the number of disk blocks used per directory or file |
| **dumbplot** | **plot**(1G) | graphics filters for various plotters |
| **dumpkeys** | **loadkeys**(1) | load and dump keyboard translation tables |

| | | |
|---|---|---|
| **e** | **ex**(1) | line editor |
| **echo** | **echo**(1V) | echo arguments to the standard output |
| **ed** | **ed**(1) | basic line editor |
| **edit** | **ex**(1) | line editor |
| **egrep** | **grep**(1V) | search a file for a string or regular expression |
| **eject** | **eject**(1) | eject media device from drive |
| **else** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **enablenumlock** | **enablenumlock**(1) | enable or disable the numlock key |
| **end** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **endif** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **endsw** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **enroll** | **xsend**(1) | send or receive secret mail |
| **env** | **env**(1) | obtain or alter environment variables |
| **eqn** | **eqn**(1) | typeset mathematics |
| **error** | **error**(1) | categorize compiler error messages, insert at source file lines |
| **eval** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **ex** | **ex**(1) | line editor |
| **exec** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **exit** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **expand** | **expand**(1) | expand TAB characters to SPACE characters, and vice versa |
| **expr** | **expr**(1V) | evaluate expressions as logical, arithmetic, or string |
| **false** | **true**(1) | provide truth values |
| **fdformat** | **fdformat**(1) | format diskettes |
| **fg** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **fgrep** | **grep**(1V) | search a file for a string or regular expression |
| **file** | **file**(1) | determine the type of a file by examining its contents |
| **find** | **find**(1) | find files by name, or by other characteristics |
| **finger** | **finger**(1) | display information about users |
| **fmt** | **fmt**(1) | simple text and mail-message formatters |
| **fmt_mail** | **fmt**(1) | simple text and mail-message formatters |
| **fold** | **fold**(1) | fold long lines for display |
| **fontedit** | **fontedit**(1) | a vfont screen-font editor |
| **foption** | **foption**(1) | determine available floating-point code generation options |
| **foreach** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **from** | **from**(1) | display the sender and date of newly-arrived mail messages |
| **ftp** | **ftp**(1C) | file transfer program |
| **gcore** | **gcore**(1) | get core images of running processes |
| **get** | **sccs-get**(1) | retrieve a version of an SCCS file |
| **get_alarm** | **set_alarm**(1) | SunView programmable alarms |
| **getoptcvt** | **getopts**(1) | parse command options in shell scripts |
| **getopt** | **getopt**(1V) | parse command options in shell scripts |
| **getopts** | **getopts**(1) | parse command options in shell scripts |
| **get_selection** | **get_selection**(1) | copy contents of SunView selection to the standard output |
| **gfxtool** | **gfxtool**(1) | run graphics programs in a SunView window |
| **gigiplot** | **plot**(1G) | graphics filters for various plotters |
| **glob** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **goto** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **gprof** | **gprof**(1) | display call-graph profile data |
| **graph** | **graph**(1G) | draw a graph |
| **grep** | **grep**(1V) | search a file for a string or regular expression |
| **groups** | **groups**(1) | display a user's group memberships |
| **hashcheck** | **spell**(1) | report spelling errors |
| **hashmake** | **spell**(1) | report spelling errors |

| | | |
|---|---|---|
| **hashstat** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **head** | **head**(1) | display first few lines of specified files |
| **help** | **sccs-help**(1) | help regarding SCCS error or warning messages |
| **help_viewer** | **help_viewer**(1) | provide help with SunView applications and desktop |
| **history** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **hostid** | **hostid**(1) | print the numeric identifier of the current host |
| **hostname** | **hostname**(1) | set or print name of current host system |
| **hpplot** | **plot**(1G) | graphics filters for various plotters |
| **i386** | **machid**(1) | return a true exit status if the processor is of the indicated type |
| **iAPX286** | **machid**(1) | return a true exit status if the processor is of the indicated type |
| **iconedit** | **iconedit**(1) | create and edit images for icons, cursors and panel items |
| **id** | **id**(1V) | print the user name and ID, and group name and ID |
| **if** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **implot** | **plot**(1G) | graphics filters for various plotters |
| **indent** | **indent**(1) | indent and format a C program source file |
| **indentpro_to_defaults** | **defaultsedit**(1) | edit default settings for SunView utilities |
| **indxbib** | **indxbib**(1) | create an inverted index to a bibliographic database |
| **inline** | **inline**(1) | in-line procedure call expander |
| **input_from_defaults** | **defaultsedit**(1) | edit default settings for SunView utilities |
| **input_from_defaults** | **input_from_defaults**(1) | update the current state of the mouse and keyboard |
| **insert_brackets** | **textedit_filters**(1) | filters provided with **textedit**(1) |
| **install** | **install**(1) | install files |
| **ipcrm** | **ipcrm**(1) | remove a message queue, semaphore set, or shared memory ID |
| **ipcs** | **ipcs**(1) | report interprocess communication facilities status |
| **jobs** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **join** | **join**(1) | relational database operator |
| **keylogin** | **keylogin**(1) | decrypt and store secret key |
| **keylogout** | **keylogout**(1) | delete stored secret key |
| **kill** | **kill**(1) | send a signal to a process, or terminate a process |
| **label** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **last** | **last**(1) | indicate last logins by user or terminal |
| **lastcomm** | **lastcomm**(1) | show the last commands executed, in reverse order |
| **ld** | **ld**(1) | link editor, dynamic link editor |
| **ldd** | **ldd**(1) | list dynamic dependencies |
| **ld.so** | **ld**(1) | link editor, dynamic link editor |
| **leave** | **leave**(1) | remind you when you have to leave |
| **lex** | **lex**(1) | lexical analysis program generator |
| **limit** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **line** | **line**(1) | read one line |
| **lint** | **lint**(1V) | a C program verifier |
| **ln** | **ln**(1V) | make hard or symbolic links to files |
| **load** | **load**(1) | load clusters |
| **loadc** | **load**(1) | load clusters |
| **loadkeys** | **loadkeys**(1) | load and dump keyboard translation tables |
| **lockscreen_default** | **defaultsedit**(1) | edit default settings for SunView utilities |
| **lockscreen_default** | **lockscreen**(1) | maintain SunView context and prevent unauthorized access |
| **lockscreen** | **lockscreen**(1) | maintain SunView context and prevent unauthorized access |
| **logger** | **logger**(1) | add entries to the system log |
| **login** | **login**(1) | log in to the system |
| **logname** | **logname**(1) | get the name by which you logged in |
| **logout** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **look** | **look**(1) | find words in the system dictionary or lines in a sorted list |
| **lookbib** | **lookbib**(1) | find references in a bibliographic database |

| | | |
|---|---|---|
| **lorder** | **lorder**(1) | find an ordering relation for an object library |
| **lp** | **lp**(1) | send/cancel requests to a printer |
| **lpq** | **lpq**(1) | display the queue of printer jobs |
| **lpr** | **lpr**(1) | send a job to the printer |
| **lprm** | **lprm**(1) | remove jobs from the printer queue |
| **lpstat** | **lpstat**(1) | display the printer status information |
| **lptest** | **lptest**(1) | generate lineprinter ripple pattern |
| **ls** | **ls**(1V) | list the contents of a directory |
| **lsw** | **lsw**(1) | list TFS whiteout entries |
| **m4** | **m4**(1V) | macro language processor |
| **m68k** | **machid**(1) | return a true exit status if the processor is of the indicated type |
| **mach** | **mach**(1) | display the processor type of the current host |
| **machid** | **machid**(1) | return a true exit status if the processor is of the indicated type |
| **Mail** | **mail**(1) | read or send mail messages |
| **mailrc_to_defaults** | **defaultsedit**(1) | edit default settings for SunView utilities |
| **mailtool** | **mailtool**(1) | SunView interface for the mail program |
| **make** | **make**(1) | maintain, update, and regenerate related programs and files |
| **man** | **man**(1) | display reference manual pages |
| **mesg** | **mesg**(1) | permit or deny messages on the terminal |
| **mkdir** | **mkdir**(1) | make a directory |
| **mkstr** | **mkstr**(1) | create an error message file by massaging C source files |
| **more** | **more**(1) | browse or page through a text file |
| **mt** | **mt**(1) | magnetic tape control |
| **mv** | **mv**(1) | move or rename files |
| **nawk** | **nawk**(1) | pattern scanning and processing language |
| **neqn** | **eqn**(1) | typeset mathematics |
| **newgrp** | **newgrp**(1) | log in to a new group |
| **nice** | **nice**(1) | run a command at low priority |
| **nl** | **nl**(1V) | line numbering filter |
| **nm** | **nm**(1) | print symbol name list |
| **nohup** | **nohup**(1V) | run a command immune to hangups and quits |
| **notify** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **nroff** | **nroff**(1) | format documents for display or line-printer |
| **objdump** | **objdump**(1) | dump selected parts of a COFF object file |
| **od** | **od**(1V) | octal, decimal, hexadecimal, and ascii dump |
| **old-ccat** | **old-compact**(1) | compress and uncompress files, and cat them |
| **old-compact** | **old-compact**(1) | compress and uncompress files, and cat them |
| **old-eyacc** | **old-eyacc**(1) | modified yacc allowing much improved error recovery |
| **old-filemerge** | **old-filemerge**(1) | window-based file comparison and merging program |
| **old-make** | **old-make**(1) | maintain, update, and regenerate groups of programs |
| **old-prmail** | **old-prmail**(1) | display waiting mail |
| **old-pti** | **old-pti**(1) | phototypesetter interpreter |
| **old-setkeys** | **old-setkeys**(1) | modify interpretation of the keyboard |
| **old-sun3cvt** | **old-sun3cvt**(1) | convert Sun-2 system executables to Sun-3 system executables |
| **old-syslog** | **old-syslog**(1) | make a system log entry |
| **old-uncompact** | **old-compact**(1) | compress and uncompress files, and cat them |
| **old-vc** | **old-vc**(1) | version control |
| **on** | **on**(1C) | execute command on a remote system with local environment |
| **onintr** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **organizer** | **organizer**(1) | file and directory manager |
| **overview** | **overview**(1) | run a program from SunView that takes over the screen |
| **pack** | **pack**(1V) | compress and expand files |
| **page** | **more**(1) | browse or page through a text file |

| | | |
|---|---|---|
| **pagesize** | **pagesize**(1) | display the size of a page of memory |
| **passwd** | **passwd**(1) | change local or NIS password information |
| **paste** | **paste**(1V) | join corresponding lines of files, subsequent lines of one |
| **pax** | **pax**(1V) | portable archive exchange |
| **paxcpio** | **paxcpio**(1V) | copy file archives in and out |
| **pcat** | **pack**(1V) | compress and expand files |
| **pdp11** | **machid**(1) | return a true exit status if the processor is of the indicated type |
| **perfmeter** | **perfmeter**(1) | display system performance values in a meter or strip chart |
| **pg** | **pg**(1V) | page through a file on a soft-copy terminal |
| **plot** | **plot**(1G) | graphics filters for various plotters |
| **popd** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **pr** | **pr**(1V) | prepare file(s) for printing, perhaps in multiple columns |
| **printenv** | **printenv**(1) | display environment variables currently set |
| **prof** | **prof**(1) | display profile data |
| **prs** | **sccs-prs**(1) | display selected portions of an SCCS history |
| **prt** | **sccs-prt**(1) | display delta table information from an SCCS file |
| **ps** | **ps**(1) | display the status of current processes |
| **ptx** | **ptx**(1) | generate a permuted index |
| **pushd** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **pwd** | **pwd**(1) | display the pathname of the current working directory |
| **quota** | **quota**(1) | display a user's disk quota and usage |
| **ranlib** | **ranlib**(1) | convert archives to random libraries |
| **rasfilter8to1** | **rasfilter8to1**(1) | convert an 8-bit deep rasterfile to a 1-bit deep rasterfile |
| **rastrepl** | **rastrepl**(1) | magnify a raster image by a factor of two |
| **rcp** | **rcp**(1C) | remote file copy |
| **rdist** | **rdist**(1) | remote file distribution program |
| **red** | **ed**(1) | basic line editor |
| **refer** | **refer**(1) | expand and insert references from a bibliographic database |
| **rehash** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **remove_brackets** | **textedit_filters**(1) | filters provided with **textedit**(1) |
| **repeat** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **reset** | **tset**(1) | establish or restore terminal characteristics |
| **rev** | **rev**(1) | reverse the order of characters in each line |
| **ring_alarm** | **set_alarm**(1) | SunView programmable alarms |
| **rlogin** | **rlogin**(1C) | remote login |
| **rm** | **rm**(1) | remove (unlink) files or directories |
| **rmdel** | **sccs-rmdel**(1) | remove a delta from an SCCS file |
| **rmdir** | **rm**(1) | remove (unlink) files or directories |
| **roffbib** | **roffbib**(1) | format and print a bibliographic database |
| **rpcgen** | **rpcgen**(1) | RPC protocol compiler |
| **rsh** | **rsh**(1C) | remote shell |
| **rup** | **rup**(1C) | show host status of local machines (RPC version) |
| **ruptime** | **ruptime**(1C) | show host status of local machines |
| **rusers** | **rusers**(1C) | who's logged in on local machines (RPC version) |
| **rwall** | **rwall**(1C) | write to all users over a network |
| **rwho** | **rwho**(1C) | who's logged in on local machines |
| **sact** | **sccs-sact**(1) | show editing activity status of an SCCS file |
| **sccs** | **sccs**(1) | front end for the Source Code Control System (SCCS) |
| **sccs-admin** | **sccs-admin**(1) | create and administer SCCS history files |
| **sccs-cdc** | **sccs-cdc**(1) | change the delta commentary of an SCCS delta |
| **sccs-comb** | **sccs-comb**(1) | combine SCCS deltas |
| **sccs-delta** | **sccs-delta**(1) | make a delta to an SCCS file |
| **sccsdiff** | **sccs-sccsdiff**(1) | compare two versions of an SCCS file |

| | | |
|---|---|---|
| **sccs-get** | **sccs-get**(1) | retrieve a version of an SCCS file |
| **sccs-help** | **sccs-help**(1) | ask for help regarding SCCS error or warning messages |
| **sccs-prs** | **sccs-prs**(1) | display selected portions of an SCCS history |
| **sccs-prt** | **sccs-prt**(1) | display delta table information from an SCCS file |
| **sccs-rmdel** | **sccs-rmdel**(1) | remove a delta from an SCCS file |
| **sccs-sact** | **sccs-sact**(1) | show editing activity status of an SCCS file |
| **sccs-sccsdiff** | **sccs-sccsdiff**(1) | compare two versions of an SCCS file |
| **sccs-unget** | **sccs-unget**(1) | undo a previous get of an SCCS file |
| **sccs-val** | **sccs-val**(1) | validate an SCCS file |
| **screenblank** | **screenblank**(1) | turn off the screen when the mouse and keyboard are idle |
| **screendump** | **screendump**(1) | dump a frame-buffer image to a file |
| **screenload** | **screenload**(1) | load a frame-buffer image from a file |
| **script** | **script**(1) | make typescript of a terminal session |
| **scrolldefaults** | **defaultsedit**(1) | edit default settings for SunView utilities |
| **sdiff** | **sdiff**(1V) | contrast two text files by displaying them side-by-side |
| **sed** | **sed**(1V) | stream editor |
| **selection_svc** | **selection_svc**(1) | SunView selection service |
| **set_alarm** | **set_alarm**(1) | SunView programmable alarms |
| **setenv** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **set** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **sh** | **sh**(1) | standard UNIX system shell and command-level language |
| **shelltool** | **shelltool**(1) | run a shell in a SunView terminal window |
| **shift** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **shift_lines** | **textedit_filters**(1) | filters provided with **textedit**(1) |
| **size** | **size**(1) | display the size of an object file |
| **sleep** | **sleep**(1) | suspend execution for a specified interval |
| **snap** | **snap**(1) | SunView application for system and network administration |
| **soelim** | **soelim**(1) | resolve and eliminate .so requests from nroff or troff input |
| **sort** | **sort**(1V) | sort and collate lines |
| **sortbib** | **sortbib**(1) | sort a bibliographic database |
| **source** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **sparc** | **machid**(1) | return a true exit status if the processor is of indicated type |
| **spell** | **spell**(1) | report spelling errors |
| **spellin** | **spell**(1) | report spelling errors |
| **spline** | **spline**(1G) | interpolate smooth curve |
| **split** | **split**(1) | split a file into pieces |
| **stop** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **strings** | **strings**(1) | find printable strings in an object file or binary |
| **strip** | **strip**(1) | remove symbols and relocation bits from an object file |
| **stty** | **stty**(1V) | set or alter the options for a terminal |
| **stty_from_defaults** | **defaultsedit**(1) | edit default settings for SunView utilities |
| **stty_from_defaults** | **stty_from_defaults**(1) | set terminal editing characters from the defaults database |
| **su** | **su**(1V) | super-user, temporarily switch to a new user ID |
| **sum** | **sum**(1V) | calculate a checksum for a file |
| **sun** | **machid**(1) | return a true exit status if the processor is of indicated type |
| **sunview** | **sunview**(1) | the SunView window environment |
| **suspend** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **sv_acquire** | **sv_acquire**(1) | change owner, group, mode of window devices |
| **sv_release** | **sv_acquire**(1) | change owner, group, mode of window devices |
| **swin** | **swin**(1) | set or get SunView user input options |
| **switcher** | **switcher**(1) | switch between multiple desktops on the same physical screen |
| **switch** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **symorder** | **symorder**(1) | rearrange a list of symbols |

| | | |
|---|---|---|
| **sync** | **sync**(1) | update the super block; force changed blocks to the disk |
| **sysex** | **sysex**(1) | start the system exerciser |
| **syswait** | **syswait**(1) | execute a command, suspending termination until user input |
| **t300** | **plot**(1G) | graphics filters for various plotters |
| **t300s** | **plot**(1G) | graphics filters for various plotters |
| **t4013** | **plot**(1G) | graphics filters for various plotters |
| **t450** | **plot**(1G) | graphics filters for various plotters |
| **tabs** | **tabs**(1V) | set tab stops on a terminal |
| **tail** | **tail**(1) | display the last part of a file |
| **talk** | **talk**(1) | talk to another user |
| **tar** | **tar**(1) | create tape archives, and add or extract files |
| **tbl** | **tbl**(1) | format tables for nroff or troff |
| **tcopy** | **tcopy**(1) | copy a magnetic tape |
| **tcov** | **tcov**(1) | construct test coverage analysis |
| **tee** | **tee**(1) | replicate the standard output |
| **tek** | **plot**(1G) | graphics filters for various plotters |
| **tektool** | **tektool**(1) | SunView Tektronix 4014 terminal-emulator window |
| **telnet** | **telnet**(1C) | user interface to a remote system using the TELNET protocol |
| **test** | **test**(1V) | return true or false according to a conditional expression |
| **textedit** | **textedit**(1) | SunView window- and mouse-based text editor |
| **textedit_filters** | **textedit_filters**(1) | filters provided with **textedit**(1) |
| **tftp** | **tftp**(1C) | trivial file transfer program |
| **then** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **time** | **time**(1V) | time a command |
| **tip** | **tip**(1C) | connect to remote system |
| **toolplaces** | **toolplaces**(1) | display current window locations, sizes, and other attributes |
| **touch** | **touch**(1V) | update the access and modification times of a file |
| **tput** | **tput**(1V) | initialize a terminal or query the terminfo database |
| **tr** | **tr**(1V) | translate characters |
| **trace** | **trace**(1) | trace system calls and signals |
| **traffic** | **traffic**(1C) | SunView program to display Ethernet traffic |
| **troff** | **troff**(1) | typeset or format documents |
| **true** | **true**(1) | provide truth values |
| **tset** | **tset**(1) | establish or restore terminal characteristics |
| **tsort** | **tsort**(1) | topological sort |
| **tty** | **tty**(1) | display the name of the terminal |
| **u3b** | **machid**(1) | return a true exit status if the processor is of the indicated type |
| **u3b2** | **machid**(1) | return a true exit status if the processor is of the indicated type |
| **u3b5** | **machid**(1) | return a true exit status if the processor is of the indicated type |
| **u3b15** | **machid**(1) | return a true exit status if the processor is of the indicated type |
| **ul** | **ul**(1) | do underlining |
| **umask** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **unalias** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **uname** | **uname**(1) | display the name of the current system |
| **uncompress** | **compress**(1) | compress or expand files, display expanded contents |
| **unexpand** | **expand**(1) | expand TAB characters to SPACE characters, and vice versa |
| **unget** | **sccs-unget**(1) | undo a previous get of an SCCS file |
| **unhash** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **unifdef** | **unifdef**(1) | resolve and remove ifdef'ed lines from cpp input |
| **uniq** | **uniq**(1) | remove or report adjacent duplicate lines |
| **units** | **units**(1) | conversion program |
| **unix2dos** | **unix2dos**(1) | convert text file from ISO format to DOS format |
| **unlimit** | **csh**(1) | C shell built-in commands, see **csh**(1) |

| | | |
|---|---|---|
| **unload** | **unload**(1) | unload optional clusters |
| **unloadc** | **unload**(1) | unload optional clusters |
| **unpack** | **pack**(1V) | compress and expand files |
| **unset** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **unsetenv** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **unwhiteout** | **unwhiteout**(1) | remove a TFS whiteout entry |
| **uptime** | **uptime**(1) | show how long the system has been up |
| **users** | **users**(1) | display a compact list of users logged in |
| **ustar** | **ustar**(1V) | process tape archives |
| **uucp** | **uucp**(1C) | system to system copy |
| **uudecode** | **uuencode**(1C) | encode a binary file, or decode its ASCII representation |
| **uuencode** | **uuencode**(1C) | encode a binary file, or decode its ASCII representation |
| **uulog** | **uucp**(1C) | system to system copy |
| **uuname** | **uucp**(1C) | system to system copy |
| **uupick** | **uuto**(1C) | public system-to-system file copy |
| **uusend** | **uusend**(1C) | send a file to a remote host |
| **uustat** | **uustat**(1C) | UUCP status inquiry and job control |
| **uuto** | **uuto**(1C) | public system-to-system file copy |
| **uux** | **uux**(1C) | remote system command execution |
| **vacation** | **vacation**(1) | reply to mail automatically |
| **val** | **sccs-val**(1) | validate an SCCS file |
| **vax** | **machid**(1) | return a true exit status if the processor is of the indicated type |
| **vedit** | **vi**(1) | visual display editor based on **ex**(1) |
| **vfontinfo** | **vfontinfo**(1) | inspect and print out information about fonts |
| **vgrind** | **vgrind**(1) | grind nice program listings |
| **vi** | **vi**(1) | visual display editor based on **ex**(1) |
| **view** | **vi**(1) | visual display editor based on **ex**(1) |
| **vplot** | **vplot**(1) | plot graphics for a Versatec printer |
| **vswap** | **vswap**(1) | convert a foreign font file |
| **vtroff** | **vtroff**(1) | troff to a raster plotter |
| **vwidth** | **vwidth**(1) | make a troff width table for a font |
| **w** | **w**(1) | who is logged in, and what are they doing |
| **wait** | **wait**(1) | wait for a process to finish |
| **wall** | **wall**(1) | write to all users logged in |
| **wc** | **wc**(1) | display a count of lines, words and characters |
| **what** | **what**(1) | extract SCCS version information from a file |
| **whatis** | **whatis**(1) | display a one-line summary about a keyword |
| **whereis** | **whereis**(1) | locate the binary, source, and manual page for a command |
| **which** | **which**(1) | locate a command; display its pathname or alias |
| **while** | **csh**(1) | C shell built-in commands, see **csh**(1) |
| **who** | **who**(1) | who is logged in on the system |
| **whoami** | **whoami**(1) | display the effective current username |
| **whois** | **whois**(1) | TCP/IP Internet user name directory service |
| **write** | **write**(1) | write a message to another user |
| **xargs** | **xargs**(1V) | construct the arguments list(s) and execute a command |
| **xget** | **xsend**(1) | send or receive secret mail |
| **xsend** | **xsend**(1) | send or receive secret mail |
| **xstr** | **xstr**(1) | extract strings from C programs to implement shared strings |
| **yacc** | **yacc**(1) | yet another compiler-compiler: parsing program generator |
| **yes** | **yes**(1) | be repetitively affirmative |
| **ypcat** | **ypcat**(1) | print values in a NIS data base |
| **ypmatch** | **ypmatch**(1) | print the value of one or more keys from a NIS map |
| **yppasswd** | **yppasswd**(1) | change your network password in the NIS database |

| | | |
|---|---|---|
| **ypwhich** | **ypwhich(1)** | return hostname of NIS server or map master |
| **zcat** | **compress(1)** | compress or expand files, display expanded contents |

## NAME

acctcom – search and print process accounting files

## SYNOPSIS

**acctcom** [ **–abfhikmqrtv** ] [ **–C** *sec* ] [ **–e** *time* ] [ **–E** *time* ] [ **–g** *group* ] [ **–H** *factor* ] [ **–I** *chars* ] [ **–l** *line* ] [ **–n** *pattern* ] [ **–o** *output-file* ] [ **–O** *sec* ] [ **–s** *time* ] [ **–S** *time* ] [ **–u** *user* ]

## DESCRIPTION

**acctcom** reads *filename*, the standard input, or **/var/adm/pacct**, in the form described by **acct**(5) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the **COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE(K)**, and optionally, **F** (the *fork/exec* flag: **1** for **fork( )** without **exec( )**), **STAT** (the system exit status), **HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD**, and **BLOCKS READ** (total blocks read and written).

A '**#**' is prepended to the command name if the command was executed with super-user privileges. If a process is not associated with a known terminal, a '**?**' is printed in the **TTYNAME** field.

If no *filename*s are specified, and if the standard input is associated with a terminal or **/dev/null** (as is the case when using '**&**' in the shell), **/var/adm/pacct** is read; otherwise, the standard input is read.

If any *filename* arguments are given, they are read in their respective order. Each file is normally read forward, that is, in chronological order by process completion time. The file **/var/adm/pacct** is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in **/var/adm/pacct**?.

## OPTIONS

**–a** Show some average statistics about the processes selected. The statistics will be printed after the output records.

**–b** Read backwards, showing latest commands first. This option has no effect when the standard input is read.

**–f** Print the *fork/exec* flag and system exit status columns in the output. The numeric output for this option will be in octal.

**–h** Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:
(total CPU time)/(elapsed time)

**–i** Print columns containing the I/O counts in the output.

**–k** Instead of memory size, show total kcore-minutes.

**–m** Show mean core size (the default).

**–q** Do not print any output records, just print the average statistics as with the **–a** option.

**–r** Show CPU factor (user time/(system-time + user-time).

**–t** Show separate system and user CPU times.

**–v** Exclude column headings from the output.

**–C** *sec* Show only processes with total CPU time, system plus user, exceeding *sec* seconds.

**–e** *time* Select processes existing at or before *time*.

**–E** *time* Select processes ending at or before *time*. Using the same *time* for both **–S** and **–E** shows the processes that existed at *time*.

**–g** *group* Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.

**–H** *factor* Show only processes that exceed *factor*, where factor is the "hog factor" as explained in the **–h** option above.

**-I** *chars* Show only processes transferring more characters than the cutoff number given by *chars*.

**-l** *line* Show only processes belonging to terminal **/dev/line**.

**-n** *pattern* Show only commands matching *pattern* that may be a regular expression as in **regexp**(3).

**-o** *ofile* Copy selected process records in the input data format to *ofile*; suppress standard output printing.

**-O** *sec* Show only processes with CPU system time exceeding *sec* seconds.

**-s** *time* Select processes existing at or after *time*, given in the format *hr* [ : *min* [ : *sec* ] ] .

**-S** *time* Select processes starting at or after *time*.

**-u** *user* Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a '#', which designates only those processes executed with super-user privileges, or '?', which designates only those processes associated with unknown user IDs.

**FILES**

**/etc/passwd**
**/var/adm/pacct**
**/etc/group**

**SEE ALSO**

**ps**(1), **su**(1V) **acct**(2V), **regexp**(3), **acct**(5), **utmp**(5V), **acct**(8), **acctcms**(8), **acctcon**(8), **acctmerg**(8), **acctprc**(8), **acctsh**(8), **fwtmp**(8), **runacct**(8)

**BUGS**

**acctcom** reports only on processes that have terminated; use **ps**(1) for active processes. If *time* exceeds the present time, then *time* is interpreted as occurring on the previous day.

## NAME

adb – general-purpose debugger

## SYNOPSIS

**adb** [ **–w** ] [ **–k** ] [ **–I** *dir* ] [ *objectfile* [ *corefile* ] ]

## AVAILABILITY

**adb** is available with the *Debugging* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**adb** is an interactive, general-purpose debugger. It can be used to examine files and provides a controlled environment for the execution of programs.

*objectfile* is normally an executable program file, preferably containing a symbol table. If the file does not contain a symbol table, it can still be examined, but the symbolic features of **adb** cannot be used. The default for *objectfile* is **a.out**. *corefile* is assumed to be a core image file produced after executing *objectfile*. The default for *corefile* is **core**.

## OPTIONS

**–k** Perform kernel memory mapping; should be used when *corefile* is a system crash dump or **/dev/mem**.

**–w** Create both *objectfile* and *corefile*, if necessary, and open them for reading and writing so that they can be modified using **adb**.

**–I** *dir* specifies a directory where files to be read with **$<** or **$<<** (see below) will be sought; the default is **/usr/lib/adb**.

## USAGE

Refer to **adb** in *Debugging Tools* for more complete information on how to use **adb**. Note: Some commands require that you compile progams to be debugged with the **–go** compiler flag; see **cc**(1V) for details. These commands are not currently available on Sun-4 systems; they are marked '**–go** only' below.

### Commands

**adb** reads commands from the standard input and displays responses on the standard output. It does not supply a prompt. It ignores the QUIT signal. INTERRUPT invokes the next **adb** command. **adb** generally recognizes command input of the form:

[ *address* ] [, *count* ] *command* [ **;** ]

*address* and *count* (if supplied) are expressions that result, respectively, in a new current address, and a repetition count. *command* is composed of a verb followed by a modifier or list of modifiers.

The symbol '.' represents the current location. It is initially zero. The default *count* is '1'.

*Verbs*

| | |
|---|---|
| **?** | Print locations starting at *address* in *objectfile*. |
| **/** | Print locations starting at *address* in *corefile*. |
| **=** | Print the value of *address* itself. |
| **@** | Interpret *address* as a source address. Print locations in *objectfile* or lines of source, as appropriate. **–go** only. |
| **:** | Manage a subprocess. |
| **$r** | Print names and contents of CPU registers. |
| **$R** | Print names and contents of MC68881 registers, if any. |
| **$x** | Print the names and contents of FPA registers 0 through 15, if any. |
| **$X** | Print the names and contents of FPA registers 16 through 31, if any. |
| **>** | Assign a value to a variable or register. |
| RETURN | Repeat the previous command with a *count* of 1. Increment '.'. |
| **!** | Shell escape. |

**Modifiers**

Modifiers specify the format of command output. Each modifier consists of a letter, preceded by an integer repeat count.

*Format Modifiers*

The following format modifiers apply to the commands ?, /, @, and =. To specify a format, follow the command with an optional repeat count, and the desired format letter or letters:

[*v*][[*r*]*f*...]

where *v* is one of these four command verbs, *r* is a repeat count, and *f* is one of the format letters listed below:

| | |
|---|---|
| **o** | ('.' increment: 2) Print 2 bytes in octal. |
| **O** | (4) Print 4 bytes in octal. |
| **q** | (2) Print in signed octal. |
| **Q** | (4) Print long signed octal. |
| **d** | (2) Print in decimal. |
| **D** | (4) Print long decimal. |
| **x** | (2) Print 2 bytes in hexadecimal. |
| **X** | (4) Print 4 bytes in hexadecimal. |
| **h** | (2) Print 2 bytes in hexadecimal in reverse order. Sun386i systems only. |
| **H** | (4) Print 4 bytes in hexadecimal in reverse order. Sun386i systems only. |
| **u** | (2) Print as an unsigned decimal number. |
| **U** | (4) Print long unsigned decimal. |
| **f** | (4) Print a single-precision floating-point number. |
| **F** | (8) Print a double-precision floating-point number. |
| **e** or **E** | (12) Print a 96-bit MC68881 extended-precision floating-point number. Sun-2 or Sun-3 systems only. |
| **b** | (1) Print the addressed byte in octal. |
| **B** | (1) Print the addressed byte in hexadecimal. Sun386i systems only. |
| **c** | (1) Print the addressed character. |
| **C** | (1) Print the addressed character using ^ escape convention. |
| **s** | (*n*) Print the addressed string. |
| **S** | (*n*) Print a string using the ^ escape convention. |
| **Y** | (4) Print 4 bytes in date format. |
| **i** | (*n*) Print as machine instructions. |
| **M** | (*n*) Print as machine instructions along with machine code. Sun386i systems only. |
| **z** | (*n*) Print with MC68010 machine instruction timings. Sun-2 or Sun-3 system only. |
| **I** | (0) Print the source text line specified by '.'. –go only. |
| **a** | (0) Print the value of '.' in symbolic form. |
| **p** | (4) Print the addressed value in symbolic form. |
| **A** | (0) Print the value of '.' in source-symbol form. |
| **P** | (4) Print the addressed value in source-symbol form. |
| **t** | (0) Tab to the next appropriate TAB stop. |
| **r** | (0) Print a SPACE. |
| **n** | (0) Print a NEWLINE. |
| **'...'** | (0) Print the enclosed string. |
| **^** | (0) Decrement '.'. |
| **+** | (0) Increment '.'. |
| – | (0) Decrement '.' by 1. |

*Modifiers for ? and / Only*

| | |
|---|---|
| **l** *value mask* | Apply *mask* and compare for *value*; move '.' to matching location. |
| **L** *value mask* | Apply *mask* and compare for 4-byte *value*; move '.' to matching location. |
| **w** *value* | Write the 2-byte *value* to address. |
| **W** *value* | Write the 4-byte *value* to address. |

**m** *b1 e1 f1*[ ?] Map new values for *b1, e1, f1*. If the **?** or **/** is followed by * then the second segment (*b2 ,e2 ,f2*) of the address mapping is changed.

: *Modifiers*

**b** *commands* Set breakpoint, execute *commands* when reached.

**B** *commands* Set breakpoint using source address, execute *commands* when reached. –**go** only.

**w** *commands* Set a data write breakpoint at *address* . Like **b** except that the breakpoint is hit when the program writes to *address* . Sun386i systems only.

**D** Delete breakpoint at source address. –**go** only.

**r** Run *objectfile* as a subprocess.

**c***s* The subprocess is continued with signal *s*.

**s***s* Single-step the subprocess with signal *s*.

**S***s* Single-step the subprocess with signal *s* using source lines. –**go** only.

**i** Add the signal specified by *address* to the list of signals passed directly to the subprocess.

**t** Remove the signal specified by *address* from the list implicitly passed to the subprocess.

**k** Single-step the subprocess with signal *s* using source lines. –**go** only.

**e***s* Like *s*, but steps over subroutine calls instead of into them. Sun386i systems only.

**u** Continue uplevel, stopping after the current routine has returned. Should only be given after the frame pointer has been pushed on the stack. Sun386i systems only.

**i** Add the signal specified by *address* to the list of signals passed directly to the subprocess

**t** Remove the signal specified by *address* from the list implicitly passed to the subprocess.

**k** Terminate the current subprocess, if any.

**A** Attach the process whose process ID is given by *address*. The PID is generally preceded by **0t** so that it will be interpreted in decimal. Sun386i systems only.

**R** Release (detach) the current process. Sun386i systems only.

$ *Modifiers*

*<filename* Read commands from the file *filename*.

*<<filename* Similar to <, but can be used in a file of commands without closing the file.

*>filename* Append output to *filename*, which is created if it does not exist.

**?** Print process ID, the signal which stopped the subprocess, and the registers.

**r** Print the names and contents of the general CPU registers, and the instruction addressed by **pc**.

**R** On Sun-3 systems with an MC68881 floating-point coprocessor, print the names and contents of the coprocessor's registers.

**x** On Sun-3 systems with a Floating Point Accelerator (FPA), print the names and contents of FPA floating-point registers 0 through 15. On Sun-4 systems, print the names and contents of the floating-point registers 0 through 15.

**X** On Sun-3 systems with an FPA, print the names and contents of FPA registers 16 through 31. On Sun-4 systems, print the names and contents of floating-point registers 16 through 31.

**b** Print all breakpoints and their associated counts and commands.

**c** C stack backtrace. On Sun-4 systems, it is impossible for **adb** to determine how many parameters were passed to a function. The default that **adb** chooses in a **$c** command is to show the six parameter registers. This can be overridden by appending a hexadecimal number to the **$c** command, specifying how many parameters to display. For example, the **$cf** command will print 15 parameters for each function in the stack trace.

**C** C stack backtrace with names and (32 bit) values of all automatic and static variables for each active function. (–**go** only).

**d** Set the default radix to *address* and report the new value. Note: *address* is interpreted in the (old) current radix. Thus '**10$d**' never changes the default radix.

**e** Print the names and values of external variables.

**w** Set the page width for output to *address* (default 80).

| | |
|---|---|
| **s** | Set the limit for symbol matches to *address* (default 255). |
| **o** | All integers input are regarded as octal. |
| **q** | Exit from **adb**. |
| **v** | Print all non zero variables in octal. |
| **m** | Print the address map. |
| **f** | Print a list of known source filenames. (–**go** only). |
| **p** | Print a list of known procedure names. (–**go** only). |
| **p** | (*Kernel debugging*) Change the current kernel memory mapping to map the designated **user structure** to the address given by *_u* ( *u* on Sun386i systems); this is the address of the user's **proc** structure. |
| **i** | Show which signals are passed to the subprocess with the minimum of **adb** interference. |
| **W** | Reopen *objectfile* and *corefile* for writing, as though the –**w** command-line argument had been given. |
| **l** | Set the length in bytes (1, 2, or 4) of the object referenced by *:a* and *:w* to *address*. Default is 1. Sun386i systems only. |

**Variables**

Named variables are set initially by **adb** but are not used subsequently.

| | |
|---|---|
| **0** | The last value printed. |
| **1** | The last offset part of an instruction source. |
| **2** | The previous value of variable 1. |
| **9** | The count on the last $< or $<< command. |

On entry the following are set from the system header in the *corefile* or *objectfile* as appropriate.

| | |
|---|---|
| **b** | The base address of the data segment. |
| **B** | The number of an address register that points to the FPA page. Sun-3 systems only. |
| **d** | The data segment size. |
| **e** | The entry point. |
| **F** | On Sun-3 systems, a value of '1' indicates FPA disassembly. |
| **m** | The 'magic' number (0407, 0410 or 0413). |
| **s** | The stack segment size. |
| **t** | The text segment size. |

**Expressions**

| | |
|---|---|
| . | The value of *dot*. |
| + | The value of *dot* incremented by the current increment. |
| ^ | The value of *dot* decremented by the current increment. |
| & | The last *address* typed. (In older versions of **adb**, '"' was used.) |
| *integer* | A number. The prefixes **0o** and **0O** indicate octal; **0t** and **0T**, decimal; **0x** and **0X**, hexadecimal (the default). |
| *int.frac* | A floating-point number. |
| *'cccc'* | ASCII value of up to 4 characters. |
| *<name* | The value of *name*, which is either a variable name or a register name. |
| *symbol* | A symbol in the symbol table. An initial '_' will be prepended to *symbol* if needed. Sun-2, Sun-3, and Sun-4 systems but not Sun386i systems. |
| *_symbol* | An external symbol. Sun-2, Sun-3, and Sun-4 systems but not Sun386i systems. |
| *routine.name* | The address of the variable *name* in the specified routine in the symbol table. If *name* is omitted, the address of the most recent stack frame for *routine*. |
| (*exp*) | The value of *exp*. |

*Unary Operators*

| | |
|---|---|
| ******exp* | The contents of location *exp* in *corefile*. |
| *%exp* | The contents of location *exp* in *objectfile* (In older versions of **adb**, '@' was used). |
| *–exp* | Integer negation. |
| *~exp* | Bitwise complement. |
| *#exp* | Logical negation. |

| | |
|---|---|
| ^F*exp* | (CTRL–F) Translate program address to source address. (–go only). |
| ^A*exp* | (CTRL–A) Translates source address to program address. (–go only). |
| `*name* | (Backquote) Translates procedure name to sourcefile address. (–go only). |
| *"file"* | The sourcefile address for the zero-th line of *file*. (–go only). |

*Binary Operators*

Binary operators are left associative and have lower precedence than unary operators.

| | |
|---|---|
| + | Integer addition. |
| – | Integer subtraction. |
| * | Integer multiplication. |
| % | Integer division. |
| & | Bitwise conjunction ("AND"). |
| \| | Bitwise disjunction ("OR"). |
| # | *lhs* rounded up to the next multiple of *rhs*. |

## FILES

**/usr/lib/adb**
**a.out**
**core**

## SEE ALSO

**cc**(1V), **dbx**(1), **ptrace**(2), **a.out**(5), **core**(5), **kadb**(8S)

*Debugging Tools*

## DIAGNOSTICS

**adb**, when there is no current command or format, comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

## BUGS

There does not seem to be any way to clear all breakpoints.

**adb** uses the symbolic information in an old and now obsolete format generated by the **–go** flag of **cc**(1V); it should be changed to use the new format generated by **–g**.

Since no shell is invoked to interpret the arguments of the **:r** command, the customary wild-card and variable expansions cannot occur.

Since there is little type-checking on addresses, using a sourcefile address in an inappropriate context may lead to unexpected results.

The **$c***parameter-count* command is a kluge.

## NAME

addbib – create or extend a bibliographic database

## SYNOPSIS

**addbib** [ **–a** ] [ **–p** *promptfile* ] *database*

## AVAILABILITY

**addbib** is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

When **addbib** starts up, answering **y** to the initial **Instructions?** prompt yields directions; typing **n** or RETURN skips them. **addbib** then prompts for various bibliographic fields, reads responses from the terminal, and sends output records to *database*. A null response (just RETURN) means to leave out that field. A '–' (minus sign) means to go back to the previous field. A trailing backslash allows a field to be continued on the next line. The repeating **Continue?** prompt allows the user either to resume by typing **y** or RETURN, to quit the current session by typing **n** or **q**, or to edit *database* with any system editor (**vi**(1), **ex**(1), **ed**(1)).

## OPTIONS

**–a** Suppress prompting for an abstract; asking for an abstract is the default. Abstracts are ended with a CTRL–D.

**–p** *promptfile*
Use a new prompting skeleton, defined in *promptfile*. This file should contain prompt strings, a TAB, and the key-letters to be written to the *database*.

## USAGE

### Bibliography Key Letters

The most common key-letters and their meanings are given below. **addbib** insulates you from these key-letters, since it gives you prompts in English, but if you edit the bibliography file later on, you will need to know this information.

| | |
|---|---|
| **%A** | Author's name |
| **%B** | Book containing article referenced |
| **%C** | City (place of publication) |
| **%D** | Date of publication |
| **%E** | Editor of book containing article referenced |
| **%F** | Footnote number or label (supplied by **refer**) |
| **%G** | Government order number |
| **%H** | Header commentary, printed before reference |
| **%I** | Issuer (publisher) |
| **%J** | Journal containing article |
| **%K** | Keywords to use in locating reference |
| **%L** | Label field used by **–k** option of **refer** |
| **%M** | Bell Labs Memorandum (undefined) |
| **%N** | Number within volume |
| **%O** | Other commentary, printed at end of reference |
| **%P** | Page number(s) |
| **%Q** | Corporate or Foreign Author (unreversed) |
| **%R** | Report, paper, or thesis (unpublished) |

**%S** Series title

**%T** Title of article or book

**%V** Volume number

**%X** Abstract — used by **roffbib**, not by **refer**

**%Y,Z** Ignored by **refer**

**EXAMPLE**

Except for A, each field should be given just once. Only relevant fields should be supplied.

**%A** **Mark Twain**
**%T** **Life on the Mississippi**
**%I** **Penguin Books**
**%C** **New York**
**%D** **1978**

**SEE ALSO**

**ed**(1), **ex**(1), **indxbib**(1), **lookbib**(1), **refer**(1), **roffbib**(1), **sortbib**(1), **vi**(1)

**refer** in *Formatting Documents*

## NAME

adjacentscreens – connect multiple screens to SunView window driver

## SYNOPSIS

**adjacentscreens** [ **–c** | **–m** ] *center-screen* [ **–l** | **–r** | **–t** | **–b** *side-screen* ] ... **–x**

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**adjacentscreens** tells the window-driver's mouse-pointer tracking mechanism how to move between screens that contain windows. Note: **sunview(1)** must be running on all screens before **adjacentscreens** is used. Once properly notified using **adjacentscreens**, the mouse pointer slides from one screen to another as you move the pointer past the appropriate edge of a screen.

## OPTIONS

**–c** *center-screen*

**–m** *center-screen*

*center-screen* is the name of a frame buffer device, such as **/dev/fb**; all other physical screen-positions are given relative to this reference point. The **–c** or –m flag is optional. If omitted, the first argument is taken as *center-screen*. If no further arguments are given, *center-screen* has no neighbors.

**–l** *side-screen*

**–r** *side-screen*

**–t** *side-screen*

**–b** *side-screen*

**–l** *side-screen*

*side-screen* is also a frame buffer device name, such as **/dev/cgone**. The **–l** flag indicates that *side-screen* is to the left of *center-screen*; **–r** indicates that it is to the right. **–t** indicates that *side-screen* is on top of (above) *center-screen*; **–b** indicates that it is below. Each neighboring screen can be specified as an option on the command line.

**–x** Suppress the normal notification to a *side-screen* pointer tracker that *center-screen* is its only neighbor. This option is useful if you have a large number of screens or want exotic inter-window pointer movements.

## EXAMPLE

The following command:

**example% adjacentscreens /dev/fb –r /dev/cgone**

sets up pointer tracking so that the pointer slides from a monochrome screen (**/dev/fb**) to a color screen (**/dev/cgone**) when the pointer moves off the right hand edge of the monochrome display. Similarly, the pointer slides from the color screen to the monochrome screen when the pointer moves off the color screen's left edge.

## FILES

**/dev/fb**
**/dev/cgone**

## SEE ALSO

**sunview(1)**, **switcher(1)**

## BUGS

Window systems on all screens must be initialized before running **adjacentscreens**.

**NAME**

apropos – locate commands by keyword lookup

**SYNOPSIS**

**apropos** *keyword...*

**DESCRIPTION**

**apropos** displays the man page name, section number, and a short description for each man page whose NAME line contains *keyword*. This information is contained in the **/usr/man/whatis** database created by **catman**(8). If **catman**(8) was not run, or was run with the **–n** option, **apropos** fails. Each word is considered separately and the case of letters is ignored. Words which are part of other words are considered; for example, when looking for 'compile', **apropos** finds all instances of 'compiler' also.

**apropos** is actually just the **–k** option to the **man**(1) command.

Try

**example% apropos password**

and

**example% apropos editor**

If the line starts '*filename*(*section*) ...' you can do '**man** *section filename*' to display the man page for *filename*. Try

**example% apropos format**

and then

**example% man 3s printf**

to get the manual page on the subroutine **printf**( ).

**FILES**

**/usr/man/whatis** data base

**SEE ALSO**

**man**(1), **whatis**(1), **catman**(8)

**DIAGNOSTICS**

**/usr/man/whatis: No such file or directory**

This database does not exist. **catman**(8) must be run to create it.

## NAME

ar – create library archives, and add or extract files

## SYNOPSIS

**ar d | m | p | q | r | t | x** [ [ **clouv** ] [ **abi** *position-name* ] ] *archive* [ *member-file*... ]

## SYSTEM V SYNOPSIS

**/usr/5bin/ar** [–] **d | m | p | q | r | t | x** [ [ **clouvs** ] [ **abi** *position-name* ] ] *archive* [ *member-file*... ]

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**ar** maintains groups of files combined into a single archive file. An archive file comprises a set of member files and header information for each file. The archive header and the headers for the file consist of printable characters (assuming that the names of the files in the archive contain printable characters), and are in a format portable across all machines. This format is described in detail in **ar**(5)). If an archive is composed of printable files, with printable file names, the entire archive is printable.

**ar** is normally used to create and update library files used by the link editor **ld**(1), but can be used for any similar purpose.

*archive* is the name of the archive file. *member-file* is a member file contained in the archive. If this argument is omitted, the command applies to all entries in the archive. Member names have a maximum of 15 characters, except on Sun386i systems, where they have a maximum of 14 characters, longer names are truncated.

## SYSTEM V DESCRIPTION

**ar** runs **ranlib** after modifying an archive, so that the symbol table member of the archive is kept up-to-date.

## OPTIONS

You must indicate only one of: **d**, **m**, **p**, **q**, **r**, **t**, or **x**, which may be followed by one or more **Modifiers.**

**d** Delete the named files from the archive file.

**m** Move the named files to the end of the archive.

**p** Print. If no names are given, all files in the archive are written to the standard output; if no names are given, only those files are written, and they are written in the order that they appear in the archive.

**q** Quick append. Append the named files to the end of the archive file without searching the archive for duplicate names. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece. If this option is used to add a member to an archive, and a member with the same name as that member already exists in the archive, the old member will not be removed; two members with the same name will exist in the archive.

**r** Replace the named files in the archive.

**t** Table of contents. If no names are given, all files in the archive are listed; if names are given, only those files are listed.

**x** Extract. If no names are given, all files in the archive are extracted into the current directory; if names are given, only those files are extracted. In neither case does **x** alter the archive file.

### Modifiers

**c** Create. Suppress the message that is produced by default when *archive* is created.

**l** Local. Place temporary files in the current working directory rather than in the default temporary directory, **/tmp.**

**o** Old date. When files are extracted with the **x** option, set the "last modified" date to the date recorded in the archive.

**u** Update. Replace only those files that have changed since they were put in the archive. Used with the **r** option.

**v** Verbose. When used with the **r**, **d**, **m**, or **q** option, give a file-by-file description of the creation of a new archive file from the old archive and the constituent files. When used with **x**, give a file-by-file description of the extraction of archive files. When used with **t**, give a long listing of information about the files. When used with **p**, write each member's name to the standard output before writing the member to the standard output.

**a** *position-name*
Place new files after *position-name* (*position-name* argument must be present). Applies only to the **r** and **m** options.

**b** *position-name*
Place new files before *position-name* (*position-name* argument must be present). Applies only to the **r** and **m** options.

**i** *position-name*
Identical to the **b** modifier.

**SYSTEM V OPTIONS**

The options may be preceded by '–'.

**Modifiers**

**s** Force the regeneration of the archive symbol table even if **ar** is not invoked with a command that will modify the archive contents.

**EXAMPLES**

Creating a new archive:

```
example% ar rcv archive file.o
a – file.o
```

Adding to an archive:

```
example% ar rav file.o archive next.c
a – next.c
```

Extracting from an archive:

```
example% ar xv archive file.o
x – file.o
example% ls file.o
file.o
```

Seeing the table of contents:

```
example% ar t archive
file.o
next.c
```

**FILES**

| | |
|---|---|
| **/tmp/v*.** | temporary files |
| **/tmp** | |

**SEE ALSO**

**ld**(1), **lorder**(1), **ranlib**(1), **ar**(5)

**BUGS**

If the same file is mentioned twice in an argument list, it is put in the archive twice.

The "last-modified" date of a file will not be altered by the **o** option unless the user is either the owner of the extracted file or the super-user.

## NAME

arch – display the architecture of the current host

## SYNOPSIS

**arch**
**arch –k**
**arch** *archname*

## DESCRIPTION

**arch** displays the application architecture of the current host system.

Sun systems can be broadly classified by their *architectures*, which define what executables will run on which machines. A distinction can be made between *kernel architecture* and *application architecture* (or, commonly, just "architecture"). Machines that run different kernels due to underlying hardware differences may be able to run the same application programs. For example, the MC68020-based Sun-3/160 system and the MC68030-based Sun-3/80 system run different SunOS kernels, but can execute the same applications. These machines could be described as having "the same application architecture" but "different kernel architectures."

Executing **arch** will display the application architecture of the machine, such as **sun3**, **sun4**, etc. All machines of the same application architecture will execute the same application programs, or user binaries.

**Current Architectures**

| Application architecture | Kernel architecture | Current Sun System Models |
|---|---|---|
| sun3 | sun3 | 3/50, 3/60, 3/75, 3/110, 3/140, 3/160, 3/180, 3/260, 3/280 |
| sun3 | sun3x | 3/80, 3/460, 3/470, 3/480 |
| sun4 | sun4 | 4/110, 4/260, 4/280, SPARCsystem 330, SPARCserver 390 |
| sun4 | sun4c | SPARCstation 1 |
| sun386 | sun386 | 386i/150, 386i/250 |

## OPTIONS

**–k** Display the kernel architecture, such as **sun3**, **sun3x**, **sun4c**, etc. This defines which specific SunOS kernel will run on the machine, and has implications only for programs that depend on the kernel explicitly (for example, **ps**(1), **vmstat**(8), etc.).

*archname* Return "true" (exit status 0) if *application* binaries for *archname* can run on the current host system, otherwise, return "false" (exit status 1). This is the preferred method for installation scripts to determine the environment of the host machine; that is, which architecture of a multi-architecture release to install on this machine. *archname* must be a valid application architecture.

## SEE ALSO

**mach**(1), **machid**(1)

*Installing SunOS 4.1*

## NAME

as – Sun-1, Sun-2 and Sun-3, Sun-4 and Sun386i assemblers

## SUN-1, SUN-2 and SUN-3 SYNOPSIS

**as** [ **–L** ] [ **–R** ] [ **–o** *objfile* ] [ **–d2** ] [ **–h** ] [ **–j** ] [ **–J** ] [ **–O** ] [ **–mc68010** ] [ **–mc68020** ] *filename*

## SUN-4 SYNOPSIS

**as** [ **–L** ] [ **–R** ] [ **–o** *objfile* ] [ **–O**[*n*] ] [ **–P** [ [ **–I***path* ] [ **–D***name* ] [ **–D***name=def* ] [ **–U***name* ] ] ... ]
[ **–S**[**C**] ] *filename* ...

## Sun386i SYNOPSIS

**as** [ **–k** ] [ **–o** *objfile* ] [ **–R** ] [ **–V** ] [ **–i386** ]

## DESCRIPTION

**as** translates the assembly source file, *filename* into an executable object file, *objfile*. The Sun-4 assembler recognizes the filename argument '–' as the standard input.

All undefined symbols in the assembly are treated as global.

The output of the assembly is left in the file *objfile*.

## OPTIONS

The following options are common to all Sun architectures. Options for specific Sun architectures are listed below.

**–L** Save defined labels beginning with an **L**, which are normally discarded to save space in the resultant symbol table. The compilers generate many such temporary labels.

**–R** Make the initialized data segment read-only by concatenating it to the text segment. This eliminates the need to run editor scripts on assembly code to make initialized data read-only and shared.

**–o** *objfile*
The next argument is taken as the name of the object file to be produced. If the **–o** flag is not used, the object file is named **a.out**.

### Sun-1, Sun-2 and Sun-3 Options

**–d2** Instruction offsets that involve forward or external references, and with unspecified size, are two bytes long. (See also the **–j** option.)

**–h** Suppress span-dependent instruction calculations. Restrict branches to medium length. Force calls to take the most general form. This option is used when the assembly must be minimized, even at the expense of program size and run-time performance. It results in a smaller and faster program than one produced by the **–J** option, but some very large programs may be unable to use it due to the limits of medium-length branches.

**–j** Use short (pc-relative) branches to resolve jump and jump-to-subroutine instructions to external routines. This is for compact programs for which the **-d2** option is inappropriate due to large-program relocation.

**–J** Suppress span-dependent instruction calculations and force branches and calls to take the most general form. This is useful when assembly time must be minimized, even at the expense of program size and run-time performance.

**–O** Perform span-dependent instruction resolution over entire files rather than just individual procedures.

### Sun-4 Options

**–O**[*n*] Enable peephole optimization corresponding to optimization level *n* (1 if *n* not specified) of the Sun high-level language compilers. This option can be used safely only when assembling code produced by a Sun compiler.

**–P** Run **cpp**(1), the C preprocessor, on the files being assembled. The preprocessor is run separately on each input file, not on their concatenation. The preprocessor output is passed to the assembler.

**-I***path*
**-D***name*
**-D***name=def*
**-U***name*

When **-P** is in effect, these **cpp**(1) options are passed to the C preprocessor, without interpretation by **as**. Otherwise, they are ignored.

**-S[C]** Produce a disassembly of the emitted code to the standard output. This is most useful in combination with the **-O** option, to review optimized code. Adding the character C to the option prevents comment lines from appearing in the output.

**Sun386i Options**

**-k** Create position independent code. Called by **cc -pic**.

**-V** Write the version number of the assembler being run on the standard error output.

**-i386** Confirm that this output is intended for an 80386 processor.

## FILES

**/tmp/as*** default temporary file

## SEE ALSO

**adb**(1), **cpp**(1), **dbx**(1), **ld**(1), **a.out**(5) **coff**(5)

*Sun-4 Assembly Language Reference*
*Sun-3 Assembly Language Reference*

## BUGS

The Sun Pascal compiler qualifies a nested procedure name by chaining the names of the enclosing procedures. This sometimes results in names long enough to abort the assembler, which currently limits identifiers to 512 characters (the Sun-4 assembler does not have this restriction).

## Sun386i WARNINGS

There can be only one forward-reference to a symbol per arithmetic expression.

## NAME

at, batch – execute a command or script at a specified time

## SYNOPSIS

**at** [ **–csm** ] [ **–q***queue* ] *time* [ *date* ] [ + *increment* ] [ *script* ]
**at –r** *jobs...*
**at –l** [ *jobs...* ]

**batch** [ **–csm** ] [ *script* ]

## DESCRIPTION

**at** and **batch** read commands from standard input to be executed at a later time. **at** allows you to specify when the commands should be executed, while jobs queued with **batch** will execute as soon as the system load level permits. *script* is the name of a file to be used as command input for the Bourne shell, **sh**(1), the C shell, **csh**(1), or an arbitrary shell specified by the SHELL environment variable. If *script* is omitted, command input is accepted from the standard input.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, and **umask**(2V) are retained when the commands are executed. Open file descriptors, traps, and priority are lost.

Users are permitted to use **at** if their name appears in the file **/var/spool/cron/at.allow**. If that file does not exist, the file **/var/spool/cron/at.deny** is checked to determine if the user should be denied access to **at**. If neither file exists, only the super-user is allowed to submit a job. If **at.deny** is empty, global usage is permitted. The allow/deny files consist of one user name per line.

The *time* may be specified as 1, 2, or 4 digits. One and two digit numbers are taken to be hours, four digits to be hours and minutes. The time may alternately be specified as two numbers separated by a colon, meaning *hour*:*minute*. A suffix **am** or **pm** may be appended; otherwise a 24-hour clock time is understood. The suffix **zulu** may be used to indicate GMT. The special names **noon**, **midnight**, **now**, and **next** are also recognized.

An optional *date* may be specified as either a month name followed by a day number (and possibly year number preceded by an optional comma) or a day of the week (fully spelled or abbreviated to three characters). Two special "days", **today** and **tomorrow** are recognized. If no *date* is given, **today** is assumed if the given hour is greater than the current hour and **tomorrow** is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed.

The optional *increment* is simply a number suffixed by one of the following: **minutes**, **hours**, **days**, **weeks**, **months**, or **years**. (The singular form is also accepted.)

Thus legitimate commands include:

**at 0815am Jan 24**
**at 8:15am Jan 24**
**at now + 1 day**
**at 5 pm Friday**

**at** and **batch** write the job number and schedule time to standard error.

**batch** submits batch jobs to queue **b**; this "batch" queue is for jobs to be run as soon as possible. A job submitted to **b** is scheduled to run immediately and its arguments will not be interpreted as *time*, *date*, or + *increment*.

**batch** is similar to '**at now**', but does not, for example, go into the same queue or respond with the error message '**too late**'.

## OPTIONS

**–c** C shell. **csh** is used to execute *script*.

**–s** Standard (Bourne) shell. **sh** is used to execute the job. By default, the SHELL environment variable determines which shell to use.

**-m** Mail. Send mail after the job has been run, even if the job completes successfully.

**-q***queue* Submit the job in queue *queue* rather than the default queue **a**. The valid *queue*s are **a** through **z**. **batch** submits jobs in queue **b**. Queue **c** is reserved for **cron**(8) and jobs cannot be submitted to that queue.

**-r** *jobs* ... Remove the specified *jobs* previously scheduled by **at** or **batch**. The job numbers are the numbers of the jobs given to you previously by the **at** or **batch** commands. You can only remove your own jobs unless you are the super-user.

**-l** [*jobs* ...]
If *jobs* is specified, print the queue entry for those *jobs*; if *jobs* is not specified, print the queue entries for all jobs for the user.

## ENVIRONMENT

If neither **at.allow** nor **at.deny** exists, only the super-user is allowed to submit a job. If **at.deny** is present, but empty, global usage is permitted. The allow/deny files consist of one user name per line.

## EXAMPLES

Unless a *script* is specified, the **at** and **batch** commands read from standard input the commands to be executed at a later time. **sh** and **csh** provide different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

**batch**
**nroff** *filename* > *outfile*
CTRL-D (hold down 'control' and depress 'D')

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

**batch <<!**
**nroff** *filename* **2>&1 >** *outfile* **| mail** *loginid*
**!**

To have a job reschedule itself, invoke **at** from within the shell procedure, by including code similar to the following within the shell file:

**at 1900 thursday next week** *shellfile*

## FILES

| | |
|---|---|
| **/var/spool/cron** | main cron directory |
| **/var/spool/cron/at.allow** | list of allowed users |
| **/var/spool/cron/at.deny** | list of denied users |
| **/var/spool/cron/atjobs** | spool area |

## SEE ALSO

**atq**(1), **atrm**(1), **csh**(1), **kill**(1), **mail**(1), **nice**(1), **ps**(1), **sh**(1), **umask**(2V), **cron**(8)

## DIAGNOSTICS

Complaints about various syntax errors and times out of range.

## BUGS

If the system crashes, mail stating that the job was not completed is not sent to the user.

Shell interpreter specifiers (such as, **!/bin/csh**) in the beginning of *script* are ignored.

**NAME**

atq – display the queue of jobs to be run at specified times

**SYNOPSIS**

**atq** [ **–c** ] [ **–n** ] *username*...

**DESCRIPTION**

**atq** prints the queue of jobs, created with the **at**(1) command, that are waiting to be run at later date.

With no flags, the queue is sorted in chronological order of execution.

If no usernames are specified, the entire queue is displayed; otherwise, only those jobs belonging to the named users are displayed.

**OPTIONS**

**–c** Creation time. Sorted the queue by the time that the **at** command was given, the most recently created job first.

**–n** Number of jobs. Print the total number of jobs currently in the queue. Do not list them.

**FILES**

**/var/spool/cron** spool area

**SEE ALSO**

**at**(1), **atrm**(1), **cron**(8)

**NAME**

atrm – remove jobs spooled by at or batch

**SYNOPSIS**

**atrm** [ **–fi** ] [ – ] [ *job-number* ] ... [ *username* ] ...

**DESCRIPTION**

**atrm** removes delayed-execution jobs that were created with the **at**(1) command. The list of jobs can be displayed by **atq**(1).

**atrm** removes each *job-number* you specify, and/or all jobs belonging to *username*, provided that you own the indicated jobs.

Jobs belonging to other users can only be removed by the super-user.

**OPTIONS**

**–f** Force. All information regarding the removal of the specified jobs is suppressed.

**–i** Interactive. **atrm** asks if a job should be removed; a response of **y** verifies that the job is to be removed.

– Remove jobs that were queued by the current user. If invoked by the super-user, the entire queue will be flushed.

**FILES**

**/var/spool/cron** spool area

**SEE ALSO**

**at**(1), **atq**(1), **cron**(8)

## NAME

awk – pattern scanning and processing language

## SYNOPSIS

**awk** [ **–f** *program-file* ] [ **–F***c* ] [ *program* ] [ *variable=value ...* ] [ *filename...* ]

## DESCRIPTION

**awk** scans each of its input *filenames* for lines that match any of a set of patterns specified in *program*. The input *filenames* are read in order; the standard input is read if there are no *filenames*. The filename '–' means the standard input.

The set of patterns may either appear literally on the command line as *program*, or, by using the '**–f** *program-file*' option, the set of patterns may be in a *program-file*; a *program-file* of '–' means the standard input. If the *program* is specified on the command line, it should be enclosed in single quotes (') to protect it from the shell.

**awk** variables may be set on the command line using arguments of the form *variable=value*. This sets the **awk** variable *variable* to *value* before the first record of the next *filename* argument is read.

With each pattern in *program* there can be an associated action that will be performed when a line of a *filename* matches the pattern. See the discussion below for the format of input lines and the **awk** language. Each line in each input *filename* is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

## OPTIONS

**–f** *program-file*
: Use the contents of *program-file* as the source for the *program*.

**–F***c*
: Use the character *c* as the field separator (FS) character. See the discussion of FS below.

## USAGE

### Input Lines

An input line is made up of fields separated by white space. The field separator can be changed by using FS — see **Special Variable Names** below. Fields are denoted **$1**, **$2**, and so forth. **$0** refers to the entire line.

### Pattern-action Statements

A pattern-action statement has the form

*pattern* { *action* }

A missing *action* means copy the line to the output; a missing *pattern* always matches.

### Action Statements

An *action* is a sequence of *statements*. A *statement* can be one of the following:

**if** ( *conditional* ) *statement* [ **else** *statement* ]
**while** ( *conditional* ) *statement*
**for** ( *expression* ; *conditional* ; *expression* ) *statement*
**break**
**continue**
{ [ *statement* ] ... }
*variable=expression*
**print** [ *expression-list* ] [ > *expression* ]
**Sprintf** *format* [ , *expression-list* ] [ > *expression* ]
**next** skip remaining patterns on this input line
**exit** skip the rest of the input

### Format of the awk Language

*statements* are terminated by semicolons, NEWLINE characters or right braces. An empty *expression-list* stands for the whole line.

*expressions* take on string or numeric values as appropriate, and are built using the operators +, −, *, /, %, and concatenation (indicated by a blank). The C operators ++ , −− , += , −= , *= , /= , and %= are also available in expressions.

*variable* may be scalars, array elements (denoted *x* [ *i* ]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric, providing a form of associative memory. String constants are quoted "... ".

The **print** statement prints its arguments on the standard output (or on a file if *>filename* is present), separated by the current output field separator, and terminated by the output record separator. The **printf** statement formats its expression list according to the format template *format* (see **printf**(3V) for a description of the formatting control characters).

**Built In Functions**

The built-in function **length** returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions **exp**, **log**, **sqrt**, and **int**, where **int** truncates its argument to an integer. '**substr**( *s*, *m*, *n* )' returns the *n*-character substring of *s* that begins at position *m*. '**sprintf** (*format*, *expression*, *expression*, ...)' formats the expressions according to the **printf** format given by *format*, and returns the resulting string.

**Patterns**

Patterns are arbitrary Boolean combinations (!, || , &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in **egrep** (see **grep**(1V)). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

*expression matchop regular-expression*
*expression relop expression*

where a *relop* is any of the six relational operators in C, and a *matchop* is either ~ (contains) or !~ (does not contain). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special pattern **BEGIN** may be used to capture control before the first input line is read, in which case **BEGIN** must be the first pattern. The special pattern **END** may be used to capture control after the last input line is read, in which case **END** must be the last pattern.

**Special Variable Names**

A single character *c* may be used to separate the fields by starting the program with

**BEGIN {FS = "***c***" }**

or by using the **−F***c* option.

Other variable names with special meanings include **NF**, the number of fields in the current record; **NR**, the ordinal number of the current record; **FILENAME**, the name of the current input file; **OFS**, the output field separator (default blank); **ORS**, the output record separator (default NEWLINE); and **OFMT**, the output format for numbers (default **%.6g**).

**EXAMPLES**

Print lines longer than 72 characters:

**length > 72**

Print first two fields in opposite order:

**{ print $2, $1 }**

Add up first column, print sum and average:

```
{ s += $1 }
END     { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

**SEE ALSO**

**grep**(1V), **lex**(1), **sed**(1V), **printf**(3V)

*Editing Text Files*

A. V. Aho, B. W. Kerninghan, P. J. Weinberger, *The AWK Programming Language* Addison-Wesley, 1988.

**NOTES**

The **awk** command is not changed to support 8-bit symbol names, as this would produce **awk** source code that is not portable between systems.

**BUGS**

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

There is no escape sequence that prints a double-quote. A workaround is to use the **sprintf** (see **printf**(3V)) function to store the character into a variable by its ASCII sequence.

```
dq = sprintf("%c", 34)
```

Syntax errors result in the cryptic message '**awk: bailing out near line 1**'.

**NAME**

banner – display a string in large letters

**SYNOPSIS**

**/usr/5bin/banner** *strings*

**AVAILABILITY**

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**banner** prints its arguments (each up to 10 characters long) in large letters on the standard output.

**SEE ALSO**

**echo**(1V)

**BUGS**

When **banner** is used with characters from the top half of the Latin-1 character set (see **iso_8859_1**(7)), each character is mapped to a best approximation for that character based on the available ASCII character shapes (for example, diacritical marks are removed).

## NAME

bar – create tape archives, and add or extract files

## SYNOPSIS

**bar** [ – ] **crxtu** [ **014578feovwbXlFmhpBisHSUZRTIN** ] [ *bar-file* ] [ *blocksize* ] [ *exclude-file* ] [ *string* ] [ *target_directory* ] [ *user_id* ] [ *include-file* ] *filename1* ... [ –**C** *dir filename* ... ] ...

## DESCRIPTION

**bar** archives and extracts multiple files onto a single **bar**, file archive, called a *bar-file*. It is quite similar to **tar**(5), but it has additional function modifiers, can read and write multiple volumes, and writes and reads a format that is incompatible with **tar** (see **bar**(5)). A bar-file is usually a magnetic tape, but it can be any file. **bar**'s actions are controlled by the first argument, the *key*, a string of characters containing exactly one function letter from the set **rxtuc**, and one or more of the optional function modifiers listed below. Other arguments to **bar** are file or directory names that specify which files to archive or extract. In all cases, the appearance of a directory name refers recursively to the files and subdirectories of that directory.

## FUNCTION LETTERS

**c** Create a new *bar-file* and write the named files onto it.

**r** Write the named files on the end of the *bar-file*. Note: this option *does not work* with quarter-inch archive tapes.

**x** Extract the named files from the *bar-file*. If a named file matches a directory with contents written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *filename* arguments are given, all files in the archive are extracted. Note: if multiple entries specifying the same file are on the tape, the last one overwrites all earlier versions.

**t** List the table of contents of the *bar-file*.

**u** Add the named files to the *bar-file* if they are not there or if they have been modified since they were last archived. Note: this option *does not work* with quarter-inch archive tapes.

## FUNCTION MODIFIERS

**014578**

Select an alternate drive on which the tape is mounted. The numbers **2**, **3**, **6**, and **9** do not specify valid drives. The default is **/dev/rmt8**.

**f** Use the next argument as the name of the *bar-file*. If **f** is omitted, use the device indicated by the **TAPE** environment variable, if set. Otherwise, use **/dev/rmt8** by default. If *bar-file* is given as '–', **bar** writes to the standard output or reads from the standard input, whichever is appropriate. Thus, **bar** can be used as the head or tail of a filter chain. **bar** can also be used to copy hierarchies with the command:

**example% cd fromdir; bar cf – . | (cd todir; bar xfBp –)**

**e** If any unexpected errors occur **bar** exits immediately with a positive exit status.

**o** Suppress information specifying owner and modes of directories which **bar** normally places in the archive. Such information makes former versions of **bar** generate an error message like:

**filename***: cannot create*

when they encounter it.

**v** Verbose. Normally **bar** does its work silently; the **v** option displays the name of each file **bar** treats, preceded by the function letter. When used with the **t** function, **v** displays the *bar-file* entries in a form similar to '**ls –l** '.

**w** Wait for user confirmation before taking the specified action. If you use **w**, **bar** displays the action to be taken followed by the file name, and then waits for a **y** response to proceed. No action is taken on the named file if you type anything other than a line beginning with **y**.

**b** Use the next argument as the blocking factor for tape records. The default blocking factor is 20 blocks. The block size is determined automatically when reading tapes (key letters **x** and **t**). This determination of the blocking factor may be fooled when reading from a pipe or a socket (see the **B** key letter below). The maximum blocking factor is determined only by the amount of memory available to **bar** when it is run. Larger blocking factors result in better throughput, longer blocks on nine-track tapes, and better media utilization.

**X** Use the next argument as a file containing a list of named files (or directories) to be excluded from the *bar-file* when using the key letters '**c**', '**x**', or '**t**'. Multiple **X** arguments may be used, with one *exclude-file* per argument.

**l** Display error messages if all links to archived files cannot be resolved. If **l** is not used, no error messages are printed.

**F** With one **F** argument specified, exclude all directories named SCCS from *bar-file*. With two arguments **FF**, exclude all directories named SCCS, all files with **.o** as their suffix, and all files named **errs**, **core**, and **a.out**.

**m** Do not extract modification times of extracted files. The modification time will be the time of extraction.

**h** Follow symbolic links as if they were normal files or directories. Normally, **bar** does not follow symbolic links.

**p** Restore the named files to their original modes, ignoring the present **umask**(2V). Setuid and sticky information are also extracted if you are the super-user. This option is only useful with the **x** key letter.

**B** Force **bar** to perform multiple reads (if necessary) so as to read exactly enough bytes to fill a block. This option exists so that **bar** can work across the Ethernet, since pipes and sockets return partial blocks even when more data is coming.

**i** Ignore directory checksum errors.

**s** Force the ownership of extracted files to match the **bar** process's effective user ID and group ID.

**H** The string of up to 128 characters is to be used as a volume header ID. A volume header is written to each volume of the archive when the **c** function letter is specified. See **bar**(5) for the volume header's format.

Use of the **H** function modifier when creating an archive allows **bar** to read volumes out of sequence. When extracting a file that spans volumes, **bar** will identify the tape(s) it needs to extract the entire file. If the wrong volume is inserted, **bar** issues a warning and prompts again for the correct volume.

**S** Place files specified for extraction in this target directory when used with the **x** function letter.

**U** Specify the user ID in the volume header when creating archive, when the **H** function modifier is used. If the **c** function letter is specified and a volume header exists, **bar** will verify that the user ids match before overwriting *bar-file* if the **N** modifier is specified.

**Z** Specify compression. **bar** will compress files when used with the **c** function letter and will decompress files when used with the **x** function letter. **bar** will neither compress a compressed file, nor decompress a decompressed file.

**R** Read the volume header and print the information to stdout.

**N** See if the user owns the media (uid matches that in the bar header) before overwriting bar-file with the **C** key word.

**T** When using the **x** or **t** function letters, terminate the search of the media after all the files specified are extracted (for **x**) or listed (for **t**).

**I** Use the next argument as a file containing a list of named files, one per line, to be included in the **bar** archive. The include file expects filenames to be followed by a semicolon and newline character.

In the case where excluded files (see **X** flag) also exist, excluded files take precedence over all included files. So, if a file is specified in both the include and exclude files (or on the command line), it will be excluded.

OPTIONS

**–C** dir filename

In a **c** (**create**) or **r** (**replace**) operation, **bar** performs a **chdir** (see **csh**(1)) to that directory before interpreting filename. This allows multiple directories not related by a close common parent to be archived using short relative path names. For example, to archive files from **/usr/include** and from **/etc**, one might use:

**example% bar c –C /usr include –C /etc .**

If you get a table of contents from the resulting **bar-file**, you will see something like:
**include/**
**include/a.out.h**
*and all the other files in* **/usr/include . . ./chown**
*and all the other files in* **/etc**

Note: the **–C** option only applies to *one* following directory name and *one* following file name.

EXAMPLES

Here is a simple example using **bar** to create an archive of your home directory on a tape mounted on drive **/dev/rmt0**:

```
example% cd
example% bar cvf /dev/rmt0 .
messages
```

The **c** option means create the archive; the **v** option makes **bar** tell you what it's doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (**/dev/rmt0** in this example).

Here is another example: **/dev/rmt0**:

```
example% cd
example% bar cvfH /dev/rmt0 "THIS IS MY HEADER" .
messages
```

As in the first example, the **c** option means create the archive; the **v** option makes **bar** tell you what it's doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (**/dev/rmt0** in this example). The **H** option says to use the string **"THIS IS MY HEADER"** as the ID field in the volume header.

Now you can read the table of contents from the archive like this:

```
example% bar tvf /dev/rmt0
(access  user-id/group-id    size  mod. date              filename)
rw-r--r-- 1677/40            2123  Nov 7 18:15:1985       ./archive/test.c
...
example%
```

You can extract files from the archive like this:

**example% bar xvf /dev/rmt0**
*messages*

If there are multiple archive files on a tape, each is separated from the following one by an EOF marker. **bar** does not read the EOF mark on the tape after it finishes reading an archive file because **bar** looks for a special header to decide when it has reached the end of the archive. Now if you try to use **bar** to read the next archive file from the tape, **bar** does not know enough to skip over the EOF mark and tries to read the EOF mark as an archive instead. The result of this is an error message from **bar** to the effect:

**bar: blocksize=0**

This means that to read another archive from the tape, you must skip over the EOF marker before starting another **bar** command. You can accomplish this using the **mt** command, as shown in the example below. Assume that you are reading from **/dev/nrmt0**.

**example% bar xvfp /dev/nrmt0** *read first archive from tape*
*messages*
**example% mt fsf 1** *skip over the end-of-file marker*
**example% bar xvfp /dev/nrmt0** *read second archive from tape*
*messages*
**example%**

Finally, here is an example using **bar** to transfer files across the Ethernet. First, here is how to archive files from the local machine (**example**) to a tape on a remote system (**host**):

**example% bar cvfb – 20** *filenames* **| rsh host dd of=/dev/rmt0 obs=20b**
*messages*
**example%**

In the example above, we are *creating* a *bar-file* with the **c** key letter, asking for *verbose* output from **bar** with the **v** option, specifying the name of the output *bar-file* using the **f** option (the standard output is where the *bar-file* appears, as indicated by the – sign), and specifying the blocksize (20) with the **b** option. If you want to change the blocksize, you must change the blocksize arguments both on the **bar** command *and* on the **dd** command.

Now, here is how to use **bar** to get files from a tape on the remote system back to the local system:

**example% rsh –n host dd if=/dev/rmt0 bs=20b | bar xvBfb – 20** *filenames*
*messages*
**example%**

In the example above, we are *extracting* from the *bar-file* with the **x** key letter, asking for *verbose output from* **bar** with the **v** option, telling **bar** it is reading from a pipe with the **B** option, specifying the name of the input *bar-file* using the **f** option (the standard input is where the *bar-file* appears, as indicated by the '–' sign), and specifying the blocksize (20) with the **b** option.

**FILES**

| | |
|---|---|
| **/dev/rmt?** | half-inch magnetic tape interface |
| **/dev/rar?** | quarter-inch magnetic tape interface |
| **/dev/rst?** | SCSI tape interface |
| **/tmp/bar*** | |

**ENVIRONMENT**

**TAPE** If specified, in the environment, the value of **TAPE** indicates the default tape device.

NOTES

**bar** will handle multiple volumes gracefully. If a tape error is encountered, **bar** issues a message on the standard error requesting a new volume. The presence of a new volume is confirmed when **bar** reads a line beginning with **Y** or **y** on the standard input; a line beginning with **N** or **n** aborts the archive; with any other character **bar** reissues the prompt.

SEE ALSO

**cpio**(1), **mt**(1), **umask**(2V), **bar**(5), **tar**(5), **dump**(8), **restore**(8)

BUGS

Neither the **r** option nor the **u** option can be used with quarter-inch archive tapes, since these tape drives cannot backspace.

There is no way to ask for the *n*th occurrence of a file.

The **u** option can be slow.

There is no way selectively to follow symbolic links.

When extracting tapes created with the **r** or **u** options, directory modification times may not be set correctly.

Files with names longer than 100 characters cannot be processed.

Filename substitution wildcards do not work for extracting files from the archive. To get around this, use a command of the form:

**bar xvf... /dev/rst0 'bar tf... /dev/rst0 | grep '***pattern* **''**

If you specify '–' as the target file and the archive spans volumes, the request for a new volume may get lost.

**NAME**

basename, dirname – display portions of pathnames and filenames

**SYNOPSIS**

**basename** *string* [ *suffix* ]
**dirname** *string*

**AVAILABILITY**

The **dirname** command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**basename** deletes any prefix ending in / and the *suffix*, if present in *string*. It directs the result to the standard output, and is normally used inside substitution marks (` `) within shell procedures.

**dirname** delivers all but the last level of the path name in *string*.

**EXAMPLES**

This shell procedure invoked with the argument **/usr/src/bin/cat.c** compiles the named file and moves the output to **cat** in the current directory:

```
cc $1
mv a.out `basename $1 .c`
```

The following example will set the shell variable NAME to **/usr/src/cmd**:

```
NAME=‘dirname /usr/src/cmd/cat.c‘
```

**SEE ALSO**

**sh**(1)

## NAME

bc – arbitrary-precision arithmetic language

## SYNOPSIS

**bc** [ **–c** ] [ **–l** ] [ *filename*... ]

## DESCRIPTION

**bc** is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. **bc** takes input from any files given, then reads the standard input.

## OPTIONS

**–c** Compile only. **bc** is actually a preprocessor for **dc**(1), which it invokes automatically, unless the **–c** (compile only) option is present. In this case the **dc** input is sent to the standard output instead.

**–l** Is the name of an arbitrary precision math library.

## USAGE

### Comments

Enclosed in /* and */.

### Names

Simple variables: *l*, where, *l* is a lower-case letter.
Array elements: *l*[*expression*], where, *expression* is a legal **bc** expression.
The words **ibase**, **obase**, and **scale**.

### Other Operands

Arbitrarily long numbers with optional sign and decimal point.

| | |
|---|---|
| (*expression*) | |
| **sqrt** (*expression*) | |
| **length** (*expression*) | Number of significant decimal digits |
| **scale** (*expression*) | Number of digits right of decimal point |
| *l*(*expression*, | ..., *expression*) |

### Operators

| | |
|---|---|
| + – * / % ^ | (% is remainder; ^ is exponent) |
| ++ — | (prefix and postfix; apply to names) |
| == <= >= != < > | |
| = += –= *= /= %= ^= | |

### Statements

*expression*
{*statement* ; ... ; *statement*}
where, *statement* is a legal **bc** statement.
**if** (*expression*)*statement*
**while** ( **expression** ) **statement**
**for** ( *expression* ; *expression* ; *expression* ) *statement*
**null statement**
**break**
**quit**

### Function Definitions

**define** *l* ( *l* ,..., *l* ) {
**auto** *l* ,..., *l*
*statement* ;... *statement*
**return** ( *expression* ) }

**Functions in –l Math Library**

| | |
|---|---|
| **s(***x***)** | sine |
| **c(***x***)** | cosine |
| **e(***x***)** | exponential |
| **l(***x***)** | log |
| **a(***x***)** | arctangent |
| **j(***n***,***x***)** | Bessel function |

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. Assignment to **scale** influences the number of digits to be retained on arithmetic operations in the manner of **dc**(1). Assignments to **ibase** or **obase** set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. 'Auto' variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables empty square brackets must follow the array name.

## EXAMPLES

Define a function to compute an approximate value of the exponential function:

```
scale = 20
define
e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

Print approximate values of the exponential function of the first ten integers:

```
for(i=1; i<=10; i++) e(i)
```

## FILES

| | |
|---|---|
| **/usr/lib/lib.b** | mathematical library |
| **dc**(1) | desk calculator proper |

## SEE ALSO

**dc**(1)

## BUGS

**for** statement must have all three *expression*'s.
**quit** is interpreted when read, not when executed.

**NAME**

biff – give notice of incoming mail messages

**SYNOPSIS**

**biff [ y | n ]**

**DESCRIPTION**

**biff** turns mail notification on or off for the terminal session. With no arguments, **biff** displays the current notification status for the terminal.

If notification is allowed, the terminal rings the bell and displays the header and the first few lines of each arriving mail message. **biff** operates asynchronously. For synchronized notices, use the **MAIL** variable of **sh**(1) or the **mail** variable of **csh**(1).

A '**biff y**' command can be included in your **.login** or **.profile** file for execution when you log in.

**OPTIONS**

**y** Allow mail notification for the terminal.

**n** Disable notification for the terminal.

**FILES**

**.login**
**.profile**

**SEE ALSO**

**cmdtool**(1), **csh**(1), **mail**(1), **sh**(1), **shelltool**(1), **sunview**(1), **comsat**(8C)

**BUGS**

You must have ownership the terminal to change its mail-notification status with **biff**, but windows running under **sunview**(1) are owned by the super-user. If you enable mail notification for the workstation console (in your **.login** or **.profile** file), incoming mail notices also appear on console windows running under **sunview**(1). See **shelltool**(1) or **cmdtool**(1) for details.

## NAME

bin-mail, binmail – an early program for processing mail messages

## SYNOPSIS

**/usr/bin/mail** [ **–ipq** ] [ **–f** *filename* ] *address*
**/usr/bin/mail** *recipient* ...

## DESCRIPTION

Note: This is the old version 7 UNIX system mail program. The default **mail** command, **/usr/ucb/mail** is described in **mail**(1).

**/usr/bin/mail** with no *address* prints a user's mail, message-by-message in last-in, first-out order. **/usr/bin/mail** accepts commands from the standard input to direct disposition messages.

When *addresses* are named, **/usr/bin/mail** takes the standard input up to an EOF (or a line with just '.') and routes it through the mailer daemon to each *recipient*. See **sendmail**(8) for details. The message is preceded by the sender's name and a postmark. Lines that look like postmarks are prepended with '>'. A *recipient* is a user name recognized by **login**(1), a network address or local mail alias, or a filename (see **aliases**(5) for details).

If there is any pending mail, **login** tells you there is mail when you log in. It is also possible to have the C shell, or the daemon **biff** tell you about mail that arrives while you are logged in.

To forward mail automatically, add the addresses of additional recipients to the **.forward** file in your home directory. Note: forwarding addresses must be valid, or the messages will bounce. You cannot, for instance, reroute your mail to a new host by forwarding it to your new address if it is not yet listed in the Network Information Service (NIS) aliases domain.

## OPTIONS

| | |
|---|---|
| **–i** | Ignore interrupts. |
| **–p** | Print messages without prompting for commands. Exit immediately upon receiving an interrupt. |
| **–q** | Quit immediately upon interrupt. |
| **–f** *filename* | Use *filename* as if it were the mail file. |

## USAGE

| | |
|---|---|
| **?** | Print a command summary. |
| CTRL-D | Put unexamined mail back in the mail file and quit. |
| **!***command* | Escape to the shell to do *command*. |
| **–** | Go back to previous message. |
| **+** | Go on to next message. |
| **RETURN** | Go on to next message. |
| **d** | Delete message and go on to the next. |
| **dq** | Delete message and quit. |
| **m** [ *recipient* ] | ... Mail the message to the named *recipients* (yourself is default). |
| **n** | Go on to next message. |
| **p** | Print message (again). |
| **q** | Same as EOT . |

**s** [ *filename* ] ... Save the message in the named *filename*s (**'mbox'** default). If saved successfully, remove it from the list and go on to the next message.

**w** [ *filename* ] ... Save the message, without a header, in the named *filename*s (**'mbox'** default). If saved successfully, remove it from the list and go on to the next message.

**x** Exit without changing the mail file.

**FILES**

| | |
|---|---|
| **/etc/passwd** | to identify sender and locate address |
| **/var/spool/mail/*** | incoming mail for user * |
| **/usr/ucb/mail** | routes input through daemon to *recipient*s |
| **mbox** | saved mail |
| **/tmp/ma*** | temp file |
| **/var/spool/mail/*.lock** | lock for mail directory |
| **dead.letter** | unmailable text is saved here |
| **$HOME/.forward** | list of forwarding recipients |

**SEE ALSO**

**biff**(1), **csh**(1), **des**(1), **login**(1), **mail**(1), **uucp**(1C), **uux**(1C), **write**(1), **xsend**(1), **crypt**(3), **aliases**(5), **sendmail**(8)

**BUGS**

Race conditions sometimes result in a failure to remove a lock file.

The super-user can read your mail, unless it is encrypted by **des**(1), **xsend**(1), or **crypt**(3). Even if you encrypt it, the super-user can delete it.

**NOTES**

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

**NAME**

cal – display a calendar

**SYNOPSIS**

**cal** [ [ *month* ] *year* ]

**DESCRIPTION**

**cal** displays a calendar for the specified year. If a month is also specified, a calendar for that month only is displayed. If neither is specified, a calendar for the present month is printed.

*year* can be between 1 and 9999. Be aware that **'cal 78'** refers to the early Christian era, not the 20th century. Also, the year is always considered to start in January, even though this is historically naive.

*month* is a number between 1 and 12.

The calendar produced is that for England and her colonies.

Try September 1752.

## NAME

calendar – a simple reminder service

## SYNOPSIS

**calendar** [ – ]

## DESCRIPTION

**calendar** consults the file **calendar** in the current directory and displays lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates — such as 'Dec. 7,' 'december 7,' and '12/7' — are recognized, but '7 December' or '7/12' are not. If you give the month as '*' with a date — for example, "* 1" — that day in any month will do. On weekends "tomorrow" extends through Monday.

When the optional '–' argument is present, **calendar** does its job for every user who has a file **calendar** in his login directory and sends him any positive results by **mail**(1). Normally this is done daily in the wee hours under control of **cron**(8).

The file **calendar** is first run through the C preprocessor, **cpp**(1), to include any other calendar files specified with the usual **#include** syntax. Included calendars are usually shared by all users, and maintained by the system administrator.

## FILES

| | |
|---|---|
| **~/calendar** | |
| **/usr/lib/calendar** | to figure out today's and tomorrow's dates |
| **/etc/passwd** | |
| **/tmp/cal*** | |
| **/lib/cpp** | |

## SEE ALSO

**at**(1), **cpp**(1), **grep**(1V), **mail**(1), **aliases**(5), **cron**(8)

## NOTES

The '–' argument works only on calendar files that are local to the machine; **calendar** is intended not to work on calendar files that are mounted remotely with NFS. Thus, '**calendar** –' should be run only on diskful machines where home directories exist; running it on a diskless client has no effect.

**calendar** is no longer in the default root crontab. Because of the network burden '**calendar** –' can induce, it is inadvisable in an environment running **ypbind** (see **ypserv**(8)) with a large passwd.byname map. However, if the usefulness of **calendar** outweighs the network impact, the super-user may run '**crontab –e**' to edit the root crontab. Otherwise, individual users may wish to use '**crontab –e**' to edit their own crontabs to have **cron** invoke **calendar** without the '–' argument, piping output to mail addressed to themselves.

## BUGS

**calendar** 's extended idea of "tomorrow" does not account for holidays.

Problems may occur when there is no **/etc/passwd** file on the local host.

The calendar mail will be sent to the user at the machine on which '**calendar** –' is run. If the system administrator wants the mail to be sent to another machine, mail aliases should be set up accordingly.

**NAME**

cat – concatenate and display

**SYNOPSIS**

**cat** [ – ] [ **–benstuv** ] [ *filename...* ]

**SYSTEM V SYNOPSIS**

**/usr/5bin/cat** [ – ] [ **–estuv** ] [ *filename...* ]

**AVAILABILITY**

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**cat** reads each *filename* in sequence and displays it on the standard output. Thus:

**example% cat goodies**

displays the contents of **goodies** on the standard output, and

**example% cat goodies1 goodies2 > goodies3**

concatenates the first two files and places the result on the third.

If no filename argument is given, or if the argument '–' is given, **cat** reads from the standard input. If the standard input is a terminal, input is terminated by an EOF condition.

**OPTIONS**

**–b** Number the lines, as **–n**, but omit the line numbers from blank lines.

**–e** Display non-printing characters, as **–v**, and in addition display a $ character at the end of each line.

**–n** Precede each line output with its line number.

**–s** Substitute a single blank line for multiple adjacent blank lines.

**–t** Display non-printing characters, as **–v**, and in addition display TAB characters as ^**I** (CTRL-I).

**–u** Unbuffered. If **–u** is not used, output is buffered in blocks, or line-buffered if standard output is a terminal.

**–v** Display non-printing characters (with the exception of TAB and NEWLINE characters) so that they are visible. Control characters print like ^**X** for CTRL-X; the DEL character (octal 0177) print as '^**?**'. Non-ASCII characters (with the high bit set) are displayed as **M–***x* where **M–** stands for '**meta**' and *x* is the character specified by the seven low order bits.

**SYSTEM V OPTIONS**

**–e** If the **–v** option is specified, display a $ character at the end of each line.

**–s** Suppress messages about files which cannot be opened.

**–t** If the **–v** option is specified, display TAB characters as ^**I** (CTRL-I) and FORMFEED characters as ^**L** (CTRL-L).

**–v** Display non-printing character (with the exception of TAB, NEWLINE, and FORMFEED characters) so that they are visible.

**ENVIRONMENT**

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **cat**. On entry to **cat**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

**SEE ALSO**

**cp**(1), **ex**(1), **more**(1), **pg**(1V), **pr**(1V), **tail**(1)

**NOTES**

Beware of '**cat a b > a**' and '**cat a b > b**', which destroy the input files before reading them.

**NAME**

cb – a simple C program beautifier

**SYNOPSIS**

**cb** [ **–js** ] [ **–l** *leng* ] [ *filename*... ]

**DESCRIPTION**

**cb** reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code.

**indent**(1) is preferred.

**OPTIONS**

With no options, **cb** preserves all user NEWLINE characters.

**–j** Join. Split lines are put back together.

**–s** Standard C style. Formats the code to the style of Kernighan and Ritchie in *The C Programming Language*.

**–l** *leng* Split lines longer than *leng*.

**SEE ALSO**

**indent**(1)

B.W. Kernighan and D.M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978

**BUGS**

Punctuation hidden in preprocessor statements can cause indentation errors.

## NAME

cc – C compiler

## SYNOPSIS

**cc** [ **–a** ] [ **–align** *_block* ] [ **–B***binding* ] [ **–c** ] [ **–C** ] [ **–dalign** ] [ **–dryrun** ] [ **–D***name* [**=***def* ] ] [ **–E** ] [ *float_option* ] [ **–fsingle** ] [ **–g** ] [ **–go** ] [ **–help** ] [ **–I***pathname* ] [ **–J** ] [ **–L***directory* ] [ **–M** ] [ **–misalign** ] [ **–o** *outputfile* ] [ **–O**[*level*] ] [ **–p** ] [ **–P** ] [ **–pg** ] [ **–pic** ] [ **–PIC** ] [ **–pipe** ] [ **–Qoption** *prog opt* ] [ **–Qpath** *pathname* ] [ **–Qproduce** *sourcetype* ] [ **–R** ] [ **–S** ] [ **–sb** ] [ **-target** *target_arch* ] [ **–temp=***directory* ] [ **–time** ] [ **–U***name* ] [ **–w** ] *sourcefile* ... [ **–l***library* ]

## SYSTEM V SYNOPSIS

**/usr/5bin/cc** *arguments*

**/usr/xpg2bin/cc** *arguments*

Note: *arguments* to **/usr/5bin/cc** and **/usr/xpg2bin/cc** are identical to those listed above.

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**cc** is the C compiler. It translates programs written in the C programming language into executable load modules, or into relocatable binary programs for subsequent loading with the **ld**(1) link editor.

In addition to the many options, **cc** accepts several types of filename arguments. For instance, files with names ending in **.c** are taken to be C source programs. They are compiled, and each resulting object program is placed in the current directory. The object file is named after its source file — the suffix **.o** replacing **.c** in the name of the object. In the same way, files whose names end with **.s** are taken to be assembly source programs. They are assembled, and produce **.o** files. Filenames ending in **.il** are taken to be inline expansion code template files; these are used to expand calls to selected routines in-line when code optimization is enabled. See FILES, below for a complete list of compiler-related filename suffixes.

Other arguments refer to assembler or loader options, object programs, or object libraries. Unless **–c**, **–S**, **–E** **–P** or **–Qproduce** is specified, these programs and libraries, together with the results of any specified compilations or assemblies, are loaded (in the order given) to produce an output file named **a.out**. You can specify a name for the executable by using the **–o** option.

If a single file is compiled and loaded all at once, the intermediate file is deleted.

**/usr/xpg2bin/cc** is a shell script that should be used to compile X/Open compliant applications. **/usr/5bin/cc** and **/usr/xpg2bin/cc** accept the same arguments and options as **cc**. **/usr/xpg2bin/cc** searches **/usr/xpg2include** for **#include** files before **/usr/include**, and specifies **/usr/xpg2lib/libxpg.a** as an additional static library of object-library routines.

## OPTIONS

When debugging or profiling objects are compiled using the **–g** or **–pg** options, respectively, the **ld** command for linking them should also contain the appropriate option.

See **ld**(1) for link-time options.

**–a** Insert code to count how many times each basic block is executed. Invokes a run-time recording mechanism that creates a **.d** file for every **.c** file (at normal termination). The **.d** file accumulates execution data for the corresponding source file. The **tcov**(1) utility can then be run on the source file to generate statistics about the program. Since this option entails some optimization, it is incompatible with **–g**.

**–align** *_block* Force the global uninitialized data symbol *block* to be page-aligned by increasing its size to a whole number of pages, and placing its first byte at the beginning of a page.

**–B***binding* Specify whether bindings of libraries for linking are **static** or **dynamic**, indicating whether libraries are non-shared or shared, respectively.

**–c** Suppress linking with **ld**(1) and produce a **.o** file for each source file. A single object file can be named explicitly using the **–o** option.

**–C** Prevent the C preprocessor, **cpp**(1), from removing comments.

**–dalign** (Sun-4 systems only.)
Generate double load/store instructions whenever possible for improved performance. Assumes that all double typed data are double aligned, and should not be used when correct alignment is not assured.

**–dryrun** Show but do not execute the commands constructed by the compilation driver.

**–D***name*[=*def*] Define a symbol *name* to the C preprocessor (**cpp**(1)). Equivalent to a **#define** directive in the source. If no *def* is given, *name* is defined as '**1**'.

**–E** Run the source file through **cpp**(1), the C preprocessor, only. Sends the output to the standard output, or to a file named with the **–o** option. Includes the **cpp** line numbering information. (See also, the **–P** option.)

*float_option* Floating-point code generation option. Can be one of:

**–f68881** Generate in-line code for Motorola MC68881 floating-point processor (supported only on Sun-3 systems).

**–ffpa** Generate in-line code for Sun Floating Point Accelerator (supported only on Sun-3 systems).

**–fsky** Generate in-line code for Sky floating-point processor (supported only on Sun-2 systems).

**–fsoft** Generate software floating-point calls. Supported only on Sun-2 and Sun-3 systems, for which it is the default.

**–fstore** Insure that expressions allocated to extended precision registers are rounded to storage precision whenever an assignment occurs in the source code. Only has effect when **–f68881** is specified (Sun-3 systems only).

**–fswitch** Run-time-switched floating-point calls. The compiled object code is linked at runtime to routines that support one of the above types of floating point code. This was the default in previous releases. Only for use with programs that are floating-point intensive, and must be portable to machines with various floating-point hardware options (supported only on Sun-2 and Sun-3 systems).

**–fsingle** (Sun-2, Sun-3 and Sun-4 systems)
Use single-precision arithmetic in computations involving only **float** expressions. Do not convert everything to **double**, which is the default. Note: floating-point *parameters* are still converted to double precision, and *functions* returning values still return double-precision values.

Although not standard C, certain programs run much faster using this option. Be aware that some significance can be lost due to lower-precision intermediate values.

**–g** Produce additional symbol table information for **dbx**(1) and **dbxtool**(1) and pass **–lg** option to **ld**(1) (so as to include the **g** library, that is: **/usr/lib/libg.a**). When this option is given, the **–O** and **–R** options are suppressed.

**–go** Produce additional symbol table information for **adb**(1). When this option is given, the **–O** and **–R** options are suppressed.

**–help** Display helpful information about **cc**.

**–I***pathname* Add *pathname* to the list of directories in which to search for **#include** files with relative filenames (not beginning with slash '/'). The preprocessor first searches for **#include** files in the directory containing *sourcefile*, then in directories named with **–I** options (if any), and finally, in **/usr/include.**

**–J** Generate 32-bit offsets in **switch** statement labels (supported only on Sun-2 and Sun-3 systems).

**–l***library* Link with object library *library* (for **ld**(1)). This option must follow the *sourcefile* arguments.

**–L***directory* Add *directory* to the list of directories containing object-library routines (for linking using **ld**(1).

**–M** Run only the macro preprocessor on the named C programs, requesting that it generate makefile dependencies and send the result to the standard output (see **make**(1) for details about makefiles and dependencies).

**–misalign** Generate code to allow loading and storage of misaligned data (Sun-4 systems only).

**–o** *outputfile* Name the output file *outputfile*. *outputfile* must have the appropriate suffix for the type of file to be produced by the compilation (see FILES, below). *outputfile* cannot be the same as *sourcefile* (the compiler will not overwrite the source file).

**–O**[*level*] Optimize the object code. Ignored when either **–g**, **–go**, or **–a** is used. **–O** with the *level* omitted is equivalent to **–O2**. On Sun386i systems, any level supplied is treated as level **1**. *level* is one of:

- **1** Do postpass assembly-level optimization only.
- **2** Do global optimization prior to code generation, including loop optimizations, common subexpression elimination, copy propagation, and automatic register allocation. **–O2** does not optimize references to or definitions of external or indirect variables.
- **3** Same as **–O2**, but optimize uses and definitions of external variables. **–O3** does not trace the effects of pointer assignments. Neither **–O3** nor **–O4** should be used when compiling either device drivers, or programs that modify external variables from within signal handlers.
- **4** Same as **–O3**, but trace the effects of pointer assignments.

**–p** Prepare the object code to collect data for profiling with **prof**(1). Invokes a run-time recording mechanism that produces a **mon.out** file (at normal termination).

**–P** Run the source file through **cpp**(1), the C preprocessor, only. Puts the output in a file with a **.i** suffix. Does not include **cpp**-type line number information in the output.

**–pg** Prepare the object code to collect data for profiling with **gprof**(1). Invokes a run-time recording mechanism that produces a **gmon.out** file (at normal termination).

**–pic** Produce position-independent code. Each reference to a global datum is generated as a dereference of a pointer in the global offset table. Each function call is generated in pc-relative addressing mode through a procedure linkage table. The size of the global offset table is limited to 64K on MC68000-family processors, or to 8K on SPARC processors.

**–PIC** Like **–pic**, but allows the global offset table to span the range of 32-bit addresses in those rare cases where there are too many global data objects for **–pic**.

**–pipe** Use pipes, rather than intermediate files, between **cpp**(1) and **ccom** compilation stages. Very cpu-intensive.

**–Qoption** *prog opt*

Pass the option *opt* to the program *prog*. The option must be appropriate to that program and may begin with a minus sign. *prog* can be one of: **as**, **cpp**, **inline**, or **ld**.

**–Qpath** *pathname*

Insert directory *pathname* into the compilation search path. *pathname will be searched for alternate* compilation programs, such as **cpp**(1), and **ld**(1). This path will also be searched first for certain relocatable object files that are implicitly referenced by the compiler driver, for example ***crt*.o** and **bb_link.o**.

**–Qproduce** *sourcetype*

Produce source code of the type *sourcetype*. *sourcetype* can be one of:

| | |
|---|---|
| **.c** | C source (from **bb_count**). |
| **.i** | Preprocessed C source from **cpp**(1). |
| **.o** | Object file from **as**(1). |
| **.s** | Assembler source (from **inline**, or **c2**). |

**–R** Merge data segment with text segment for **as**(1). Data initialized in the object file produced by this compilation is read-only, and (unless linked with **ld –N**) is shared between processes. Ignored when either **–g** or **–go** is used.

**–S** Do not assemble the program but produce an assembly source file.

**–sb** Generate extra symbol table information for the Sun Source Code Browser. This is an unbundled product that will be released based on 4.1.

**-target** *target_arch*

Compile object files for the specified processor architecture. Unless used in conjunction with one of the Sun Cross-Compilers, correct programs can be generated only for the architecture of the host on which the compilation is performed. *target_arch* can be one of:

| | |
|---|---|
| **sun2** | Produce object files for a Sun-2 system. |
| **sun3** | Produce object files for a Sun-3 system. |
| **sun4** | Produce object files for a Sun-4 system. |

**–temp=***directory*

Set directory for temporary files to be *directory*.

**–time** Report execution times for the various compilation passes.

**–U***name* Remove any initial definition of the **cpp**(1) symbol *name*. Inverse of the **–D** option.

**–w** Do not print warnings.

**ENVIRONMENT**

**FLOAT_OPTION** (Sun-2, Sun-3, Sun-4 systems only.) When no floating-point option is specified, the compiler uses the value of this environment variable (if set). Recognized values are: **f68881**, **ffpa**, **fsky**, **fswitch** and **fsoft**.

**FILES**

| | |
|---|---|
| **a.out** | executable output file |
| *file*.**a** | library of object files |
| *file*.**c** | C source file |
| *file*.**d** | **tcov**(1) test coverage input file (Sun-2, Sun-3, Sun-4 systems only) |
| *file*.**i** | C source file after preprocessing with **cpp**(1) |
| *file*.**il** | *inline* expansion file |
| *file*.**o** | object file |
| *file*.**s** | assembler source file |
| *file*.**S** | assembler source for **cpp**(1) |
| *file*.**tcov** | output from **tcov**(1) (Sun-2, Sun-3, Sun-4 systems only) |
| **/usr/lib/c2** | object code optimizer |

| | |
|---|---|
| **/usr/lib/ccom** | compiler |
| **/usr/lib/compile** | compiler command-line processing driver |
| **/usr/lib/cpp** | macro preprocessor |
| **/usr/lib/crt0.o** | runtime startup code |
| **/usr/lib/Fcrt1.o** | startup code for **–fsoft** option (Sun-2, Sun-3, Sun-4 systems only) |
| **/usr/lib/gcrt0.o** | startup for profiling with **gprof**(1) |
| **/usr/lib/libc.a** | standard library, see **intro**(3) |
| **/usr/lib/mcrt0.o** | startup for profiling with **prof**(1) **intro**(3) |
| **/usr/lib/Mcrt1.o** | startup code for **–f68881** option (for Sun-3 systems) |
| **/lib/optim** | Sun386i code optimizer |
| **/lib/Scrt1.o** | startup code for **–fsky** option (for Sun-2 systems) |
| **/usr/lib/Wcrt1.o** | startup code for **–ffpa** option (for Sun-3 systems) |
| **/usr/include** | standard directory for **#include** files |
| **/usr/lib/bb_link.o** | basic block counting routine |
| **/usr/lib/cg** | code generator used with **/usr/lib/iropt** |
| **/usr/lib/libc_p.a** | profiling library, see **gprof**(1) or **prof**(1) |
| **/usr/lib/inline** | inline expander of library calls |
| **/usr/lib/iropt** | intermediate representation optimizer |
| **/usr/lib/libm.a** | math library |
| **/usr/5lib/libc.a** | System V standard compatibility library, see **intro**(3) |
| **/usr/5lib/libc_p.a** | System V profiling library, see **gprof**(1) or **prof**(1) |
| **/tmp/*** | compiler temporary files |
| **/usr/xpg2include** | directory for X/Open **#include** files |
| **/usr/xpg2lib/libxpg.a** | X/Open XPG2 compatibility library |
| **/usr/xpg2lib/libxpg_p.a** | Profiled version of X/Open XPG2 compatibility library |
| **mon.out** | file produced for analysis by **prof**(1) |
| **gmon.out** | file produced for analysis by **gprof**(1) |
| **.cb** | subdirectory that holds the information generated by the **–cb** option |

## SEE ALSO

**adb**(1), **ar**(1V), **as**(1), **cflow**(1V), **cpp**(1), **ctags**(1), **cxref**(1V), **dbx**(1), **dbxtool**(1), **gprof**(1), **inline**(1), **ld**(1), **lint**(1V), **m4**(1V), **make**(1), **prof**(1), **tcov**(1), **intro**(3), **monitor**(3)

*Numerical Computation Guide*
*Programming Utilities and Libraries*
B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978

## DIAGNOSTICS

The diagnostics produced by the C compiler are intended to be self-explanatory. Occasional obscure messages may be produced by the preprocessor, assembler, or loader.

## NOTES

While the compiler allows 8-bit strings and comments, 8-bits are not allowed anywhere else. The **cc** command does not generate or support 8-bit symbol names because, until ANSI C, non-ASCII support was not expected. The ANSI C specification now suggests that string literals and comments can contain any characters from any character code set.

The following commands are affected by this lack of support for 8-bit characters: **cflow**(1V), **cpp**(1), **ctags**(1), **cxref**(1V), **dbx**(1), **lint**(1V), **m4**(1V), and **yacc**(1).

## BUGS

The program context given in syntax error messages is taken from the input text *after* the C preprocessor has performed substitutions. Therefore, error messages involving syntax errors in or near macro references or manifest constants may be misleading.

Compiling with optimization level **2** or greater may produce incorrect object code if tail-recursion elimination is applied to functions called with fewer actual parameters (arguments) than the number of formal parameters in the function's definition. Such parameter-count mismatches can be detected using **lint**(1V).

**NAME**

cd – change working directory

**SYNOPSIS**

**cd** [ *directory* ]

**DESCRIPTION**

*directory* becomes the new working directory. The process must have execute (search) permission in *directory*. If **cd** is used without arguments, it returns you to your login directory. In **csh**(1) you may specify a list of directories in which *directory* is to be sought as a subdirectory if it is not a subdirectory of the current directory; see the description of the **cdpath** variable in **csh**(1).

**SEE ALSO**

**csh**(1), **pwd**(1), **sh**(1)

## NAME

cflow – generate a flow graph for a C program

## SYNOPSIS

**cflow** [**–r**] [**–ix**] [ **–i_** ] [ **–d***num* ] *filenames*

## SYSTEM V SYNOPSIS

**cflow** [**–r**] [**–ix**] [ **–i_** ] [ **–d***num* ] *filenames*

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**cflow** analyzes a collection of C, **yacc**, **lex**, assembler, and object files and attempts to build a graph charting the external references. Files suffixed in **.y** and **.l** are run through **yacc** and **lex**, respectively; the output of **yacc** and **lex** for those files, and files suffixed in **.c**, are first run through the C preprocessor and then run through the first pass of **lint**(1V). (The **–I**, **–D**, and **–U** options of the C preprocessor are also understood.) Files suffixed in **.i** are passed directly to the first pass of **lint**. Files suffixed with **.s** are assembled. Information is extracted from the symbol tables of the output of the assembler and from files suffixed with **.o**. The output of all this non-trivial processing is collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference (that is, line) number, followed by a suitable number of tabs indicating the level. Then the name of the global (normally only a function not defined as an external or beginning with an underscore; see below for the **–i** inclusion option) a colon and its definition. For information extracted from C source, the definition consists of an abstract type declaration (for example, **char ***), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (for example, *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only < > is printed.

## SYSTEM V DESCRIPTION

The System V version of **cflow** in **/usr/5bin/cflow** makes the C preprocessor, **cpp**(1) search in **/usr/5include** for include files before it searches in **/usr/include**.

## OPTIONS

The following options are interpreted by **cflow** :

**–r** Reverse the "caller:callee" relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.

**–ix** Include external and static data symbols. The default is to include only functions in the flowgraph.

**–i_** Include names that begin with an underscore. The default is to exclude these functions (and data if **–ix** is used).

**–d***num* The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be met with contempt.

**EXAMPLES**

As an example, given the following in **file.c**:

```
int i;
main()
{
        f();
        g();
        f();
}
f()
{
        i = h();
}
```

the command:

```
cflow -ix file.c
```

produces the output

```
1       main: int(), <file.c 4>
2               f: int(), <file.c 11>
3                       h: <>
4                       i: int, <file.c 1>
5               g: <>
```

When the nesting level becomes too deep, the **-e** option of **pr**(1V) can be used to compress the tab expansion to something less than eight spaces.

**SEE ALSO**

**as**(1), **cc**(1V), **cpp**(1), **lex**(1), **lint**(1V), **nm**(1), **pr**(1V), **yacc**(1)

**DIAGNOSTICS**

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used, such as the C preprocessor.

**NOTES**

While the compiler allows 8-bit strings and comments, 8-bits are not allowed anywhere else. See **cc**(1V) for an explanation about why **cc** is not 8-bit clean.

**BUGS**

Files produced by **lex** and **yacc** cause the reordering of line number declarations which can confuse **cflow**. To get proper results, feed **cflow** the **yacc** or **lex** input.

## NAME

checknr – check nroff and troff input files; report possible errors

## SYNOPSIS

**checknr** [ **–fs** ] [ **–a** *.x1 .y1 .x2 .y2 ... .xn .yn* ] [ **–c** *.x1 .x2 .x3 ... .xn* ] [ *filename ...* ]

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**checknr** checks a list of **nroff**(1) or **troff**(1) input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. If no files are specified, **checknr** checks the standard input. Delimiters checked are:

- Font changes using \f*x* ... \fP.
- Size changes using \s*x* ... \s0.
- Macros that come in open ... close forms, for example, the .TS and .TE macros which must always come in pairs.

**checknr** knows about the **ms**(7) and **me**(7) macro packages.

**checknr** is intended to be used on documents that are prepared with **checknr** in mind. It expects a certain document writing style for \f and \s commands, in that each \f*x* must be terminated with \fP and each \s*x* must be terminated with \s0. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from **checknr**. Since it is probably better to use the \fP and \s0 forms anyway, you should think of this as a contribution to your document preparation style.

## OPTIONS

**–f** Ignore \f font changes.

**–s** Ignore \s size changes.

**–a** *.x1 .y1...*

Add pairs of macros to the list. The pairs of macros are assumed to be those (such as .DS and .DE) that should be checked for balance. The **–a** option must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair .BS and .ES, use '**–a.BS.ES**'

**–c** *.x1 ...*

Define commands which **checknr** would otherwise complain about as undefined.

## SEE ALSO

**eqn**(1), **nroff**(1), **troff**(1), **me**(7), **ms**(7)

## BUGS

There is no way to define a one-character macro name using the **–a** option.

**NAME**

chgrp – change the group ownership of a file

**SYNOPSIS**

**chgrp [ –f ] [ –R ]** *groupfilename...*

**DESCRIPTION**

**chgrp** changes the group ID (GID) of the *filenames* given as arguments to *group*. The group may be either a decimal GID or a group name found in the GID file, **/etc/group**.

You must belong to the specified group and be the owner of the file, or be the super-user.

**OPTIONS**

**–f** Force. Do not report errors.

**–R** Recursive. **chgrp** descends through the directory, and any subdirectories, setting the specified GID as it proceeds. When symbolic links are encountered, their group is changed, but they are not traversed.

**FILES**

**/etc/group**

**SEE ALSO**

**chown**(2V), **group**(5), **passwd**(5)

**NAME**

chkey – create or change encryption key

**SYNOPSIS**

**chkey**

**DESCRIPTION**

**chkey** prompts the user for their login password, and uses it to encrypt a new encryption key for the user to be stored in the **publickey**(5) database.

If the entry for **nobody** in the **/etc/publickey** database has not been commented out, **chkey** can be used to create new entries. If this entry has been commented out, **chkey** can only be used to change an existing entry.

**SEE ALSO**

**keylogin**(1), **publickey**(5), **keyserv**(8C), **newkey**(8)

**NAME**

chmod – change the permissions mode of a file

**SYNOPSIS**

**chmod** [ **–fR** ] *mode filename* ...

**SYNOPSIS**

**/usr/5bin/chmod** [ **–fR** ] *mode filename* ...

**AVAILABILITY**

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

Change the permissions (mode) of a file or files. Only the owner of a file (or the super-user) may change its mode.

The mode of each named file is changed according to *mode*, which may be absolute or symbolic, as follows.

**Absolute Modes**

An absolute *mode* is an octal number constructed from the OR of the following modes:

| | |
|---|---|
| **400** | Read by owner. |
| **200** | Write by owner. |
| **100** | Execute (search in directory) by owner. |
| **040** | Read by group. |
| **020** | Write by group. |
| **010** | Execute (search) by group. |
| **004** | Read by others. |
| **002** | Write by others. |
| **001** | Execute (search) by others. |
| **4000** | Set user ID on execution. |
| **2000** | Set group ID on execution (this bit is ignored if the file is a directory; it may be set or cleared only using symbolic mode). |
| **1000** | Sticky bit, (see **chmod**(2V) for more information). |

**Symbolic Modes**

A symbolic *mode* has the form:

[ *who* ] *op permission* [ *op permission* ] ...

*who* is a combination of:

| | |
|---|---|
| **u** | User's permissions. |
| **g** | Group permissions. |
| **o** | Others. |
| **a** | All, or **ugo**. |

If *who* is omitted, the default is **a**, but the setting of the file creation mask (see **umask** in **sh**(1) or **csh**(1) for more information) is taken into account. When *who* is omitted, **chmod** will not override the restrictions of your user mask.

*op* is one of:

| | |
|---|---|
| + | To add the *permission*. |
| – | To remove the *permission*. |
| = | To assign the permission explicitly (all other bits for that category, owner, group, or others, will be reset). |

*permission* is any combination of:

**r** Read.
**w** Write.
**x** Execute.
**X** Give execute permission if the file is a directory or if there is execute permission for one of the other user classes.
**s** Set owner or group ID. This is only useful with **u** or **g**. Also, the set group ID bit of a directory may only be modified with '+' or '–'.
**t** Set the sticky bit to save program text between processes.

The letters **u**, **g**, or **o** indicate that *permission* is to be taken from the current mode for the user-class.

Omitting *permission* is only useful with '=', to take away all permissions.

Multiple symbolic modes, separated by commas, may be given. Operations are performed in the order specified.

**SYSTEM V DESCRIPTION**

If *who* is omitted in a symbolic mode, it does not take the file creation mask into account, but acts as if *who* were **a**.

**OPTIONS**

**–f** Force. **chmod** will not complain if it fails to change the mode of a file.

**–R** Recursively descend through directory arguments, setting the mode for each file as described above. When symbolic links are encountered, their mode is not changed and they are not traversed.

**EXAMPLES**

The first example denies write permission to others, the second makes a file executable by all if it is executable by anyone:

```
example% chmod o–w file
example% chmod +X file
```

**SEE ALSO**

**csh**(1), **ls**(1V), **sh**(1), **chmod**(2V), **chown**(8)

**NAME**

clear – clear the terminal screen

**SYNOPSIS**

**clear**

**DESCRIPTION**

**clear** clears your screen if this is possible. It looks in the environment for the terminal type and then in **/etc/termcap** to figure out how to clear the screen.

**FILES**

**/etc/termcap** terminal capability data base

**NAME**

clear_colormap – clear the colormap to make console text visible

**SYNOPSIS**

**clear_colormap** [ **–no** ] [ **–f** *framebuffer* ]

**DESCRIPTION**

**clear_colormap** ensures that text displayed on the console is visible. If no options are specified it clears the frame buffer and initializes the colormap entries for the first two colors and the last color. If the frame buffer has an overlay plane it is also cleared, its colormap is initialized, and the overlay enable plane is set so that the entire overlay plane is displayed.

**OPTIONS**

**–n** Do not clear the frame buffer or overlay plane.

**–o** Do not clear the overlay plane, initialize its colormap, or modify the overlay enable plane.

**–f** *framebuffer*

Operate on frame buffer device *framebuffer* instead of the default, **/dev/fb**.

**FILES**

**/dev/fb**

**NAME**

clear_functions – reset the selection service to clear stuck function keys

**SYNOPSIS**

**clear_functions**

**DESCRIPTION**

**clear_functions** instructs the selection service that no function keys are currently depressed. It is useful in cases where erroneous programs have reported a key press but not the corresponding release. The usual symptom for this situation is that all selections are secondary (underscored rather than inverted), even though no function keys are down.

**FILES**

**/usr/bin/selection_svc**

**SEE ALSO**

*SunView User's Guide*

**NAME**

click – enable or disable the keyboard's keystroke click

**SYNOPSIS**

**click** [ **–y** ] [ **–n** ] [ **–d** *keyboard-device* ]

**DESCRIPTION**

Change the setting of the audible keyboard click. The default is no keyboard click. If you want to turn clicking on then a good place to do it is in **/etc/rc.local**.

Only keyboards that support a clicker respond to this command. At the time of this writing, the only keyboards known to have a clicker are the Sun-3 and Sun386i systems keyboards.

**OPTIONS**

**–y** Yes, enable clicking.

**–n** No, disable clicking.

**–d** *keyboard-device*
Specify the keyboard device being set. The default is **/dev/kbd**.

**FILES**

**/etc/rc.local**
**/dev/kbd**

**SEE ALSO**

**kbd**(4S)

**DIAGNOSTICS**

A short help message is printed if an unknown flag is specified or if the **–d** switch is used and the keyboard device name is not supplied. A diagnostic is printed if the keyboard device name can't be opened or is not a keyboard type device.

**BUGS**

There is no way to determine the state of the keyboard click setting.

**NAME**

clock – display the time in an icon or window

**SYNOPSIS**

**clock** [ **–frst** ] [ **–d mdyaw** ]

**AVAILABILITY**

This command is available with the *SunView User's* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**clock** is a SunView utility that displays the current time in a window. When open, **clock** shows the date and time in text format. When closed, **clock** displays a clock face.

Note: In previous releases **clock** was known as **clocktool**. In the current release, **/usr/old/clocktool** is a symbolic link to **clock**.

**OPTIONS**

**–f** Display the date and day of week on the clock face.

**–r** Use a square face with roman numerals in the iconic state. This replaces the default round clock face.

**–s** Start **clock** with the seconds turned on. By default, the clock starts with seconds turned off, and updates every minute. With seconds turned on, it updates every second, and, if iconic, displays a second hand.

**–t** Test mode — ignore the real time, and instead run in a loop continuously incrementing the time by one minute and displaying it.

**–d** Display date information in a small area just below the clock face. The date information to be displayed may include:

- **m** the month,
- **d** the day of the month (1-31),
- **y** the year,
- **a** the string **AM** or **PM**, as appropriate,
- **w** the day of the week (Sun-Sat).

There is only room for 3 of these, but any 3 may be displayed in any sequence.

**clock** also accepts all of the generic tool arguments discussed in **sunview**(1).

**USAGE**

**clock** listens for keyboard input while open. It recognizes the following characters:

**s** or **S** Enable or disable the display of seconds.

**t** or **T** Enable or disable "test" mode.

**FILES**

**/usr/lib/fonts/fixedwidthfonts/sail.r.6**
font for day-of-month clock-face display

**SEE ALSO**

**date**(1V), **sunview**(1)

**NOTES**

When the **–r** option is specified, **clock** displays the roman numeral four as **IIII** instead of **IV** to counterbalance the roman numeral eight, **VIII**.

**BUGS**

If you reset the system time, **clock** will not reflect the new time until you change its state from open to icon, or vice versa. To reset the system time, see **date**(1V).

The date display does not go well with the round clock face.

## NAME

cluster – find the Application SunOS or Developer's Toolkit optional cluster containing a file

## SYNOPSIS

**cluster** [ *filename* ]

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

**cluster** finds the optional software cluster that contains a specified file. If the specified file is contained in one of the clusters in Application SunOS or Developer's Toolkit, the name of the cluster will be printed on standard output.

Without arguments, **cluster** displays a summary of the clusters in Application SunOS and Developers Toolkit, including the load state and size of each cluster.

## EXAMPLES

To find the name of the cluster that contains the **spell** command:

```
example% cluster spell
spellcheck
example%
```

To display a summary of the clusters in Application SunOS and Developer's Toolkit:

```
% cluster
Application SunOS Clusters:
   available              cluster  size (bytes)
   ---------  -------     ----
   yes        accounting  265K
   no         advanced_admin 501K
   ...

Developer's Toolkit Clusters:
   availablecluster       size (bytes)
   ---------  -------     ----
   no         base_devel  6907K
   ...

space used by clusters: 6021K bytes
total space remaining: 20432K bytes
```

A cluster is available if it has been "loaded" using **load**(1) or if it has been "mounted" across the network.

## FILES

**/usr/lib/load/*** data files

## SEE ALSO

**load**(1), **unload**(1), **toc**(5)

*Sun386i System Setup and Maintenance*

## DIAGNOSTICS

**The file** *filename* **is not in any of the optional software clusters.**

The specified file is not part of the Application SunOS or Developer's Toolkit.

## NAME

cmdtool – run a shell (or program) using the SunView text facility

## SYNOPSIS

**cmdtool** [ **–C** ] [ **–M** *bytes* ] [ **–P** *count* ] [ *generic-tool-arguments* ] [ *program* [ *program-arguments* ] ]

## AVAILABILITY

This command is available with the *SunView User's* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**cmdtool** is the standard *SunView* support facility for shells and other programs. When invoked, **cmdtool** runs a program (usually a shell) in a text-based command subwindow. Characters typed on the keyboard are inserted at the caret. If the program is a shell, that shell accepts and runs commands in the usual manner. **cmdtool** also supports programs that perform cursor motions directly, such as vi(1).

The text of the current command line can be edited using normal **textedit**(1) functions. The command subwindow displays a log of the session, which can be scrolled through using the scrollbar (unless the command does cursor motion). This log can be edited, and saved by choosing the **'Store as New File'** item in the text facility's pop-up menu. The **'Split View'** command, also in the pop-up menu, can be used to create two or more independent views of the log.

## OPTIONS

**–C** Console **cmdtool**. Display console messages in this **cmdtool**, which might otherwise appear in unexpected places on the workstation screen. Since a **cmdtool** window can be scrolled, console error messages can be recorded for later examination.

**–M** *bytes* Set the log to wrap-around after the indicated number of *bytes*.

**–P** *count* Checkpoint the log after every set of *count* editing operations.

*generic-tool-arguments*
**cmdtool** accepts the generic tool arguments listed in **sunview**(1).

*program* [ *program-arguments* ]
If a *program* argument is present, **cmdtool** runs it and passes any remaining arguments to that *program*. If no *program* is given, **cmdtool** runs the program indicated by the **SHELL** environment variable, or **/usr/bin/sh** by default.

## USAGE

Refer to *SunView User's Guide* for details on how to use **cmdtool**.

### Defaults Options

The following options can be configured as default settings using **defaultsedit**(1).

**/Tty/Append_only_log**
When set to **TRUE** (the standard default) only command lines can be edited. **FALSE** allows the entire log to be edited. (See also the description of **Enable Edit** below.)

**/Tty/Insert_makes_caret_visible**
This entry allows you to specify the method for displaying the editing caret.

**Same_as_for_text** Use the setting specified in the defaults for the **Text** category (the standard default).

**If_auto_scroll** If the caret is showing, and an inserted NEWLINE would position it below the bottom of the screen (as determined by **/Text/Lower_context**), the text is scrolled to keep the caret showing. The number of lines scrolled is determined by the **/Text/Auto_scroll_by** default. (See **textedit**(1) for more information.)

**Always** Scroll the caret back into view whenever input would position it off the screen.

**/Tty/Checkpoint_frequency**

If set to **0** (the standard default) no checkpointing is done. For any value greater than zero, a checkpoint is made each time the indicated number editing operations has been performed since the last checkpoint. Each character typed, each **Paste**, and each **Cut** counts as an editing operation. At each checkpoint, an updated copy of the log is saved in a file with a name that is constructed by appending two percent signs (**%%**) to the name of the log file. By default, the log file has a name of the form **/tmp/tty.txt.***pid* (*pid* is the process ID number of **cmdtool**); the corresponding checkpoint file has a name of the form **/tmp/tty.txt.***nnnnnn***%%**.

**/Tty/Text_wraparound_size**

If set to **0** (the standard default) no wrap-around takes place; the log file grows without a specified limit. For values greater than zero, wrap-around occurs whenever the indicated number of characters have been written to the log since the last wrap-around. Characters that are pushed over the top are replaced by the message:

***** Text is lost because the maximum edit log size has been exceeded. *****

**/Text/Edit_back_char**

Set the character for erasing to the left of the caret. Note: in **cmdtool**, the '**stty erase**' command has no effect. Text-based tools refer only to the defaults database key settings. The default is DELETE.

**/Text/Edit_back_word**

Set the character for erasing the word to the left of the caret. The standard default is CTRL-W.

**/Text/Edit_back_line**

Set the character for erasing all characters to the left of the caret. Note: '**stty kill**' has no effect in **cmdtool**. The standard default is CTRL-U.

### The Command Subwindow

The command subwindow is based on the text facility, which is described in *SunView User's Guide*. It uses the same pop-up menu as the text facility, but with an additional pull-right '**Cmd Modes**' menu, which contains the '**Enable Editing**' and '**Disable Scrolling**' items.

Command subwindows support cursor motions, using a new **/etc/termcap** entry called **sun-cmd**. Command subwindows automatically set the TERM environment variable to **sun-cmd**. So, if you **rlogin**(1C) to a machine that does not have an entry for **sun-cmd** in its **/etc/termcap** file, the error message '**Type sun-cmd unknown**' results. To rectify this, type the command '**set TERM=sun**'. Programs written using the **curses**(3V) library package will work in a command subwindow, but programs hard-coded for **sun**-type terminals may not. When supporting a program that performs cursor motions, the command subwindow automatically takes on the characteristics of a tty subwindow (as with **shelltool**(1)). When that program terminates or sleeps, the full command subwindow functionality is restored.

**cmdtool** supports programs that use CBREAK and NO ECHO terminal modes. This support is normally invisible to the user. However, programs that use RAW mode, such as **rlogin**(1C) and **script**(1), inhibit command-line editing with the mouse. In this case, however, tty-style ERASE, word-kill and line-kill characters can still be used to edit the current command line.

### The Command Subwindow Menu

**Copy, then Paste** When there is a current selection, the entire menu item is active. The selection is copied both to the clipboard and to the location pointed to by the caret. When there is no selection, but there is text on the clipboard, only **Paste** is active. In this case, the contents of the clipboard are copied to the caret. When there is no selection and the clipboard is empty, this item is inactive. '**Copy then Paste**' is a generic text menu item. Refer to **textedit**(1) for information about other generic text menu items.

**Enable Edit**
**Disable Edit** Toggle to allow or disallow editing on the log.

**Disable Scrolling**
**Enable Scrolling** Toggle between a scrollable, editable window, or a display that supports cursor motions. Note: for well-behaved programs (such as **vi**(1)) this switching is performed automatically (so this menu item is seldom needed).

**Accelerators**

Text facility accelerators that are especially useful in command subwindows are described here. See **textedit**(1) for more information.

CTRL-RETURN Position the caret at the bottom, and scroll it into view as determined by **/Text/Lower_context**.

META-P Choose the **'Copy, then Paste'** menu item.

CAPS-lock
F1 Toggle between all-upper-case keyboard input, and mixed-case.

## ENVIRONMENT

The environment variables **LC_CTYPE, LANG,** and **LC_default** control the character classification throughout **cmdtool**. On entry to **cmdtool**, these environment variables are checked in the following order: **LC_CTYPE, LANG,** and **LC_default.** When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE.** When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## FILES

**/tmp/tty.txt.***pid* log file
**~/.textswrc**
**~/.ttyswrc**
**usr/lib/.text_extras_menu**
**/etc/termcap**

## SEE ALSO

**defaultsedit**(1), **rlogin**(1C), **script**(1), **sh**(1), **shelltool**(1), **sunview**(1), **textedit**(1), **vi**(1), **curses**(3V)

*Installing SunOS 4.1*
*SunView User's Guide*

## BUGS

Typing ahead while **cmdtool** changes between its scrollable and cursor motion modes will sometimes freeze **cmdtool .**

Full terminal emulation is not complete. Some manifestations of this deficiency are:

- File completion in the C shell does not work.
- Enhanced display of text is not supported.

**NAME**

cmp – perform a byte-by-byte comparison of two files

**SYNOPSIS**

**cmp** [ **–ls** ] *filename1 filename2* [ *skip1* ] [ *skip2* ]

**DESCRIPTION**

**cmp** compares *filename1* and *filename2*. If *filename1* is '–', the standard input is used. With no options, **cmp** makes no comment if the files are the same; if they differ, it reports the byte and line number at which the difference occurred, or, that one file is an initial subsequence of the other. *skip1* and *skip2* are initial byte offsets into *filename1* and *filename2* respectively, and may be either octal or decimal; a leading **0** denotes octal.

**OPTIONS**

**–l** Print the byte number (in decimal) and the differing bytes (in octal) for all differences between the two files.

**–s** Silent. Print nothing for differing files; set exit codes only.

**SEE ALSO**

**comm**(1), **diff**(1)

**DIAGNOSTICS**

Exit code **0** is returned for identical files, **1** for different files, and **2** for an inaccessible or missing argument, or a system error.

## NAME

col – filter reverse paper motions from nroff output for display on a terminal

## SYNOPSIS

**col** [ **–bfhp** ]

## SYSTEM V SYNOPSIS

**/usr/5bin/col** [ **–bfpx** ]

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**col** copies the standard input to the standard output and performs line overlays implied by reverse LINEFEED characters (ESC–7 in ASCII) and by forward and reverse half LINEFEED characters (ESC–9 and ESC–8). **col** is particularly useful for filtering multicolumn output made with the **.rt** command of **nroff**(1), and output resulting from use of the **tbl**(1) preprocessor.

The control characters SO (ASCII code 017), and SI (016) are assumed to start and end text in an alternate character set. The character set (primary or alternate) associated with each printing character read is remembered; on output, SO and SI characters are generated where necessary to maintain the correct treatment of each character.

All control characters are removed from the input except SPACE, BACKSPACE, TAB, RETURN, NEWLINE, ESC (033) followed by one of 7, 8, 9, SI , SO , and VT (013). This last character is an alternate form of full reverse LINEFEED, for compatibility with some other hardware conventions. All other non-printing characters are ignored.

## SYSTEM V DESCRIPTION

The System V version of **col** converts SPACE to TAB characters by default.

## OPTIONS

**–b** The output device in use is not capable of backspacing. In this case, if several characters are to appear in the same place, only the last one read will be taken.

**–f** Fine. Although **col** accepts half line motions in its input, it normally does not produce them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. The **–f** option suppresses this treatment. In this case the output from **col** may contain forward half LINEFEED characters (ESC–9), but will still never contain either kind of reverse line motion.

**–h** Convert strings of blanks to TAB characters to decrease the printing time.

**–p** Pass escape-sequences that **col** does not know about to the output, rather than stripping them out. This option is highly discouraged unless you are fully aware of the position of the escape sequences within the text.

## SYSTEM V OPTIONS

**–x** Suppress converting SPACE characters to TAB characters.

## ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **col**. On entry to **col**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## SEE ALSO

**nroff**(1), **tbl**(1), **troff**(1), **locale**(5), **iso_8859_1**(7)

**BUGS**

**col** cannot back up more than 128 lines.

At most 1600 characters, including BACKSPACE characters, are allowed on a line.

Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

**NAME**

colcrt – filter nroff output for a terminal lacking overstrike capability

**SYNOPSIS**

**colcrt** [ – ] [ **–2** ] [ *filename* ... ]

**DESCRIPTION**

**colcrt** provides virtual half and reverse LINEFEED sequences for terminals without such capability, and on which overstriking is destructive. Half LINEFEED characters and underlining (changed to dashing '–') are placed on new lines in between the normal output lines.

**OPTIONS**

– Suppress all underlining — especially useful for previewing *allboxed* tables from **tbl**(1).

**–2** Print all half LINEFEED characters, effectively double spacing the output. Normally, a minimal space output format is used which suppresses empty lines. **colcrt** never suppresses two consecutive empty lines, however. The **–2** option is useful for sending output to the line printer when the output contains superscripts and subscripts which would otherwise be invisible.

**EXAMPLE**

A typical use of **colcrt** would be

**example% tbl exum2.n | nroff –ms | colcrt – | more**

**SEE ALSO**

**col**(1V), **more**(1), **nroff**(1), **tbl**(1), **troff**(1), **ul**(1)

**BUGS**

Cannot back up more than 102 lines.

General overstriking is lost; as a special case '|' overstruck with '–' or underline becomes '+'.

Lines are trimmed to 132 characters.

Some provision should be made for processing superscripts and subscripts in documents which are already double-spaced.

## NAME

coloredit – alter color map segment

## SYNOPSIS

**coloredit**

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

**coloredit** is a SunView application for altering the colors of objects (windows) selected with the cursor. These colors may be manipulated in three ways:

- select color name from a list
- specify the hue/saturation/value for a color
- specify the red/green/blue levels for a color

## USAGE

### Window

The **coloredit** window consists of four sections.

colorlist — An interactive window that lists the available colors for selection. This list can be scrolled up and down with a scrollbar, and comprises the contents of the **.rgb** file. Select a color and the corresponding RGB and HSV value appears in the colorbar section. **coloredit** first searches the working directory for **.rgb**, your home directory, and finally, **/usr/lib** this file.

colorbars — An interactive window with two sets of three sliding bars. One set is for defining the hue saturation value of the color desired. The second set is for defining the red-green-blue ratios. The color can be specified using either set of sliding bars.

palette — A display subwindow divided horizontally into two equal parts. The top part shows the background color; the bottom, the foreground color. If the object has more than 2 colors, all the colors are shown. The background color is color Index #0. The color with the highest Index Number is the foreground color.

logo — A display subwindow containing the Sun logo in the currently-selected color.

### Command Buttons

The set of command buttons in the colorbars subwindow is as follows.

**Grab** The next object selected will have its color map segment colors displayed in the *coloredit* window. These colors may then be manipulated.

**Release** Disassociates the colormap segment and the last object with which those colors were associated. The window returns to its default state.

**Undo** Returns the colormap segment to the original colors that the object being colored had when coloring began. Returns the object to its original colors.

## FILES

**./.rgb**
**~/.rgb**
**/usr/lib/.rgb**

## SEE ALSO

**sunview**(1)

**NAME**

colrm – remove characters from specified columns within each line

**SYNOPSIS**

**colrm** [ *startcol* [ *endcol* ] ]

**DESCRIPTION**

**colrm** removes selected columns from a text file. The text is taken from standard input and copied to the standard output with the specified columns removed.

If only *startcol* is specified, the columns of each line are removed starting with *startcol* and extending to the end of the line. If both *startcol* and *endcol* are specified, all columns between *startcol* and *endcol*, inclusive, are removed.

Column numbering starts with column 1.

**SEE ALSO**

**expand**(1)

**NAME**

comm – display lines in common, and lines not in common, between two sorted lists

**SYNOPSIS**

**comm** [ **–1** | **–2** | **–3** | **–12** | **–13** | **–23** ] *filename1 filename2*

**DESCRIPTION**

**comm** reads *filename1* and *filename2*, which should be ordered in ASCII collating sequence (see **sort**(1V)), and produces three-column output when no options are specified:

- Column 1 contains lines that occur only in *filename1*.
- Column 2 contains lines only in *filename2*.
- Column 3 contains lines common to both files.

The filename '–' means the standard input.

**OPTIONS**

The following options can be used to suppress the indicated columns from display. You can specify '**–123**', but doing so suppresses all output.

**–1** Suppress column 1; omit lines only in *filename1*.

**–2** Suppress column 2; omit lines only in *filename2*.

**–3** Suppress column 3; omit lines common to both files.

**–12** Suppress columns 1 and 2; only show lines common to both files.

**–13** Suppress columns 1 and 3; only show lines in *filename2*.

**–23** Suppress columns 2 and 3; only show lines in *filename1*.

**SEE ALSO**

**cmp**(1), **diff**(1), **sort**(1V), **uniq**(1)

**BUGS**

The options *suppress*, rather than *select* the columns you indicate.

## NAME

compress, uncompress, zcat – compress or expand files, display expanded contents

## SYNOPSIS

**compress** [ **–cfv** ] [ **–b** *bits* ] [ *filename*... ]

**uncompress** [ **–cv** ] [ *filename*... ]

**zcat** [ *filename*... ]

## DESCRIPTION

**compress** reduces the size of the named files using adaptive Lempel-Ziv coding. Whenever possible, each file is replaced by one with the extension **.Z**, while keeping the same ownership modes, as well as access and modification times. If no files are specified, the standard input is compressed to the standard output.

The amount of compression obtained depends on the size of the input, the number of *bits* per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50–60%. Compression is generally much better than that achieved by Huffman coding (as used in **sys-unconfig**(8)), or adaptive Huffman coding (**old-compact**(1)), and takes less time to compute. The *bits* parameter specified during compression is encoded within the compressed file, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is subsequently allowed.

Compressed files can be restored to their original form using **uncompress**.

**zcat** produces uncompressed output on the standard output, but leaves the compressed **.Z** file intact.

## OPTIONS

**–c** Write to the standard output; no files are changed. The nondestructive behavior of **zcat** is identical to that of '**uncompress –c**'.

**–f** Force compression, even if the file does not actually shrink, or the corresponding **.Z** file already exists. Except when running in the background (under **sh**(1)), if **–f** is not given, prompt to verify whether an existing **.Z** file should be overwritten.

**–v** Verbose. Display the percentage reduction for each file compressed.

**–b** *bits* Set the upper limit (in bits) for common substring codes. *bits* must be between 9 and 16 (16 is the default).

## SEE ALSO

**ln**(1V), **old-compact**(1), **sh**(1), **sys-unconfig**(8)

*A Technique for High Performance Data Compression*, Terry A. Welch, *computer*, vol. 17, no. 6 (June 1984), pp. 8-19.

## DIAGNOSTICS

Exit status is normally 0. If the last file was not compressed because it became larger, the status is 2. If an error occurs, exit status is 1.

**Usage: compress [–fvc] [–b maxbits]** [*filename*...]
Invalid options were specified on the command line.

**Missing maxbits**
Maxbits must follow **–b** .

*filename*: **not in compressed format**
The file specified to **uncompress** has not been compressed.

*filename*: **compressed with** *xx***bits, can only handle** *yy***bits**
*filename* was compressed by a program that could deal with more *bits* than the compress code on this machine. Recompress the file with smaller *bits*.

*filename*: **already has .Z suffix -- no change**
The file is assumed to be already compressed. Rename the file and try again.

*filename*: **already exists; do you wish to overwrite (y or n)?**
Respond **y** if you want the output file to be replaced; **n** if not.

**uncompress: corrupt input**
A SIGSEGV violation was detected, which usually means that the input file is corrupted.

**Compression:** *xx.xx%*
Percentage of the input saved by compression. (Relevant only for **-v**.)

**-- not a regular file: unchanged**
When the input file is not a regular file, (such as a directory), it is left unaltered.

**-- has** *xx* **other links: unchanged**
The input file has links; it is left unchanged. See **ln**(1V) for more information.

**-- file unchanged**
No savings are achieved by compression. The input remains uncompressed.

## BUGS

Although compressed files are compatible between machines with large memory, **-b12** should be used for file transfer to architectures with a small process data space (64KB or less).

**compress** should be more flexible about the existence of the **.Z** suffix.

**NAME**

cp – copy files

**SYNOPSIS**

**cp** [ **–ip** ] *filename1 filename2*
**cp –rR** [ **–ip** ] *directory1 directory2*
**cp** [ **–iprR** ] *filename ... directory*

**DESCRIPTION**

**cp** copies the contents of *filename1* onto *filename2*. The mode and owner of *filename2* are preserved if it already existed; the mode of the source file is used otherwise. If *filename1* is a symbolic link, or a duplicate hard link, the contents of the file that the link refers to are copied; links are not preserved.

In the second form, **cp** recursively copies *directory1*, along with its contents and subdirectories, to *directory2*. If *directory2* does *not* exist, **cp** creates it and duplicates the files and subdirectories of *directory1* within it. If *directory2* does exist, **cp** makes a copy of the *directory1* directory within *directory2* (as a subdirectory), along with its files and subdirectories.

In the third form, each *filename* is copied to the indicated *directory*; the basename of the copy corresponds to that of the original. The destination *directory* must already exist for the copy to succeed.

**cp** refuses to copy a file onto itself.

**OPTIONS**

**–i** Interactive. Prompt for confirmation whenever the copy would overwrite an existing file. A **y** in answer confirms that the copy should proceed. Any other answer prevents **cp** from overwriting the file.

**–p** Preserve. Duplicate not only the contents of the original file or directory, but also the modification time and permission modes.

**–r**
**–R** Recursive. If any of the source files are directories, copy the directory along with its files (including any subdirectories and their files); the destination must be a directory.

**EXAMPLES**

To copy a file:

```
example% cp goodies goodies.old
example% ls goodies*
goodies goodies.old
```

To copy a directory, first to a new, and then to an existing destination directory:

```
example% ls ~/bkup
/usr/example/fred/bkup not found
example% cp –r ~/src ~/bkup
example% ls –R ~/bkup
x.c y.c z.sh
example% cp –r ~/src ~/bkup
example% ls –R ~/bkup
src x.c y.c z.sh

src:
x.c y.c z.sh
```

To copy a list of files to a destination directory:

```
example% cp ~/src/* /tmp
```

**SEE ALSO**

**cat**(1V), **ln**(1V), **mv**(1), **pr**(1V), **rcp**(1C), **tar**(1)

**WARNINGS**

*Beware* of a recursive copy like this:

**example% cp -r ~/src ~/src/bkup**

which keeps copying files until it fills the entire file system.

**BUGS**

**cp** copies the contents of files pointed to by symbolic links. It does *not* copy the symbolic link itself. This can lead to inconsistencies when directory hierarchies are replicated. Filenames that were linked in the original hierarchy are no longer linked in the replica. This is also true for files with multiple hard links. See **ln**(1V) for details about symbolic links and hard links. You can preserve links in replicated hierarchies by using **tar**(1) to copy them.

## NAME

cpio – copy file archives in and out

## SYNOPSIS

**cpio –o** [ **aBcv** ]
**cpio –i** [ **bBcdfmrsStuv6** ] [ *patterns* ]
**cpio –p** [ **adlmuv** ] *directory*

## DESCRIPTION

**cpio** copies files in to and out from a **cpio** copy archive. The archive (built by '**cpio –o**') contains pathname and status information, along with the contents of one or more archived files.

## OPTIONS

**–o** Copy out an archive. Read the standard input for a list of pathnames, then copy the named files to the standard output in archive form — including pathname and status information.

- **a** Reset the access times of input files after they have been copied.
- **B** Output is to be blocked at 5120 bytes to the record. This does not apply to the *pass* option. This option is only meaningful with data directed to raw magnetic devices, such as '**/dev/rmt***?*'.
- **c** Write *header* information in ASCII character form for portability.
- **v** Verbose. A list of filenames is displayed. When used with the **t** option, the table of contents looks like the output of an '**ls –l**' command (see **ls**(1V)).

**–i** Copy in an archive. Read in an archive from the standard input and extract files with names matching filename substitution *patterns*, supplied as arguments.

*patterns* are similar to those in **sh**(1) or **csh**(1), save that within **cpio**, the metacharacters '**?**', '*****' and '[ ... ]' also match the '/' (slash) character. If no *patterns* are specified, the default is * (select all files).

- **b** Swap both bytes and half-words after reading in data.
- **B** Input is to be blocked at 5120 bytes to the record. This does not apply to the *pass* option. This option is only meaningful with data received from raw magnetic devices, such as '**/dev/rmt***?*'.
- **d** Create directories as needed.
- **f** Copy in all files except those matching *patterns*.
- **m** Retain previous file modification time. This option is ineffective on directories that are being copied.
- **r** Interactively rename files. If the user types a null line, the file is skipped. May not be used with the **–p** option.
- **s** Swap bytes after reading in data.
- **S** Swap halfwords after reading in data.
- **t** Print a table of contents of the input archive. No files are created.
- **u** Copy unconditionally. Normally, an older file will not replace a newer file with the same name.
- **6** Process UNIX Version-6 files.

**–p** One pass. Copy in and out in a single operation. Destination pathnames are interpreted relative to the named *directory*.

- **l** Whenever possible, link files rather than copying them.

**EXAMPLES**

To copy the contents of a directory into an archive:

**example% ls | cpio –o > /dev/mt0**

To read a cpio archive from a tape drive:

**example% cpio -icdB < /dev/rmt0**

To duplicate the **olddir** directory hierarchy in the **newdir** directory:

**example% cd olddir**
**example% find . –depth –print | cpio –pdl newdir**

The trivial case

**example% find . –depth –print | cpio –oB >/dev/rmt0**

can be handled more efficiently by:

**example% find . –cpio /dev/rmt/0m**

**cpio** archive tapes from other sites may have bytes swapped within the archive. Although the **–is** option only swaps the data bytes and not those in the header **cpio** recognizes tapes like this and swaps the bytes in the header automatically.

**SEE ALSO**

**ar**(1V), **csh**(1), **find**(1), **ls**(1V), **sh**(1), **tar**(1), **cpio**(5)

**BUGS**

**cpio** does not support multiple volume tapes.

Pathnames are restricted to 128 characters. If there are too many unique linked files, **cpio** runs out of memory and linking information is lost thereafter. Only the super-user can copy special files.

## NAME

cpp – the C language preprocessor

## SYNOPSIS

**/usr/lib/cpp** [ **–BCHMpPRT** ] [ **–undef** ] [ **–D***name* ] [ **–D***name=def* ] [ **–I***directory* ] [ **–U***name* ] [ **–Y***directory* ] [ *input-file* [ *output-file* ] ]

## DESCRIPTION

**cpp** is the C language preprocessor. It is invoked as the first pass of any C compilation started with the **cc**(1V) command; however, **cpp** can also be used as a first-pass preprocessor for other Sun compilers.

Although **cpp** can be used as a macro processor, this is not normally recommended, as its output is geared toward that which would be acceptable as input to a compiler's second pass. Thus, the preferred way to invoke **cpp** is through the **cc**(1V) command, or some other compilation command. For general-purpose macro-processing, see **m4**(1V), and the chapter on **m4** in *Programming Utilities and Libraries.*

**cpp** optionally accepts two filenames as arguments. *input-file* and *output-file* are, respectively, the input and output files for the preprocessor. They default to the standard input and the standard output.

## OPTIONS

**–B** Support the C++ comment indicator '//'. With this indicator everything on the line after the // is treated as a comment.

**–C** Pass all comments (except those that appear on **cpp** directive lines) through the preprocessor. By default, **cpp** strips out C-style comments.

**–H** Print the pathnames of included files, one per line on the standard error.

**–M** Generate a list of makefile dependencies and write them to the standard output. This list indicates that the object file which would be generated from the input file depends on the input file as well as the include files referenced.

**–p** Use only the first eight characters to distinguish preprocessor symbols, and issue a warning if extra tokens appear at the end of a line containing a directive.

**–P** Preprocess the input without producing the line control information used by the next pass of the C compiler.

**–R** Allow recursive macros.

**–T** Use only the first eight characters for distinguishing different preprocessor names. This option is included for backward compatibility with systems which always use only the first eight characters.

**–undef** Remove initial definitions for all predefined symbols.

**–D***name*

Define *name* as 1 (one). This is the same as if a **–D***name*=**1** option appeared on the **cpp** command line, or as if a

**#define** *name* **1**

line appeared in the source file that **cpp** is processing.

**–D***name=def*

Define *name* as if by a **#define** directive. This is the same as if a

**#define** *name* **def**

line appeared in the source file that **cpp** is processing. The **–D** option has lower precedence than the **–U** option. That is, if the same name is used in both a **–U** option and a **–D** option, the name will be undefined regardless of the order of the options.

**–I***directory*

Insert *directory* into the search path for **#include** files with names not beginning with '/'. *directory* is inserted ahead of the standard list of "include" directories. Thus, **#include** files with names enclosed in double-quotes (") are searched for first in the directory of the file with the **#include** line, then in directories named with **–I** options, and lastly, in directories from the standard list. For **#include** files with names enclosed in angle-brackets (<>), the directory of the file with the **#include** line is not searched. See **Details** below for exact details of this search order.

**–U***name*

Remove any initial definition of *name*, where *name* is a symbol that is predefined by a particular preprocessor. Here is a partial list of symbols that may be predefined, depending upon the architecture of the system:

| | |
|---|---|
| Operating System: | **ibm**, **gcos**, **os**, **tss** and **unix** |
| Hardware: | **interdata**, **pdp11**, **u370**, **u3b**, **u3b2**, **u3b5**, **u3b15**, **u3b20d**, **vax**, **ns32000**, **iAPX286**, **i386**, **sparc** , and **sun** |
| UNIX system variant: | **RES**, and **RT** |
| The **lint**(1V) command: | **lint** |

The symbols **sun** and **unix** are defined for all Sun systems, as is the value returned by the **mach** command. For Sun-4 systems, this value would be **sparc**.

**–Y***directory*

Use directory *directory* in place of the standard list of directories when searching for **#include** files.

## USAGE

### Directives

All **cpp** directives start with a hash symbol (#) as the first character on a line. White space (SPACE or TAB characters) can appear after the initial # for proper indentation.

**#define** *name token-string*

Replace subsequent instances of *name* with *token-string*.

**#define** *name*(*argument* [, *argument*] ... ) *token-string*

There can be no space between *name* and the '('. Replace subsequent instances of *name*, followed by a parenthesized list of arguments, with *token-string*, where each occurrence of an *argument* in the *token-string* is replaced by the corresponding token in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, **cpp** re-starts its scan for names to expand at the beginning of the newly created *token-string*.

**#undef** *name*

Remove any definition for the symbol *name*. No additional tokens are permitted on the directive line after *name*.

**#include** "*filename*"
**#include** <*filename*>

Read in the contents of *filename* at this location. This data is processed by **cpp** as if it were part of the current file. When the <*filename*> notation is used, *filename* is only searched for in the standard "include" directories. See the **–I** and **–Y** options above for more detail. No additional tokens are permitted on the directive line after the final '"' or '>'.

**#line** *integer-constant "filename"*

Generate line control information for the next pass of the C compiler. *integer-constant* is interpreted as the line number of the next line and *filename* is interpreted as the file from where it comes. If "*filename*" is not given, the current filename is unchanged. No additional tokens are permitted on the directive line after the optional *filename*.

**#if** *constant-expression*

Subsequent lines up to the matching **#else**, **#elif** , or **#endif** directive, appear in the output only if *constant-expression* yields a nonzero value. All binary non-assignment C operators, including '&&', '||', and ',', are legal in *constant-expression*. The '**?:**' operator, and the unary '–', '**!**', and '~' operators, are also legal in *constant-expression*.

The precedence of these operators is the same as that for C. In addition, the unary operator **defined**, can be used in *constant-expression* in these two forms: '**defined** ( *name* )' or '**defined** *name*'. This allows the effect of **#ifdef** and **#ifndef** directives (described below) in the **#if** directive. Only these operators, integer constants, and names that are known by **cpp** should be used within *constant-expression*. In particular, the **size of** operator is not available.

**#ifdef** *name*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has been defined, either with a **#define** directive or a **–D** option, and in the absence of an intervening **#undef** directive. No additional tokens are permitted on the directive line after *name*.

**#ifndef** *name*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has *not* been defined, or if its definition has been removed with an **#undef** directive. No additional tokens are permitted on the directive line after *name* .

**#elif** *constant-expression*

Any number of **#elif** directives may appear between an **#if**, **#ifdef**, or **#ifndef** directive and a matching **#else** or **#endif** directive. The lines following the **#elif** directive appear in the output only if all of the following conditions hold:

- The *constant-expression* in the preceding **#if** directive evaluated to zero, the *name* in the preceding **#ifdef** is not defined, or the *name* in the preceding **#ifndef** directive *was* defined.
- The *constant-expression* in all intervening **#elif** directives evaluated to zero.
- The current *constant-expression* evaluates to non-zero.

If the *constant-expression* evaluates to non-zero, subsequent **#elif** and **#else** directives are ignored up to the matching **#endif**. Any *constant-expression* allowed in an **#if** directive is allowed in an **#elif** directive.

**#else** This inverts the sense of the conditional directive otherwise in effect. If the preceding conditional would indicate that lines are to be included, then lines between the **#else** and the matching **#endif** are ignored. If the preceding conditional indicates that lines would be ignored, subsequent lines are included in the output. Conditional directives and corresponding **#else** directives can be nested.

**#endif** End a section of lines begun by one of the conditional directives **#if**, **#ifdef**, or **#ifndef**. Each such directive must have a matching **#endif**.

**Macros**

Formal parameters for macros are recognized in **#define** directive bodies, even when they occur inside character constants and quoted strings. For instance, the output from:

```
#define abc(a) |\a|
abc(xyz)
```

is the seven characters " |'xyz|" (SPACE, vertical-bar, backquote, x, y, z, vertical-bar). Macro names are not recognized within character constants or quoted strings during the regular scan. Thus:

```
#define abc  xyz
printf("abc");
```

does not expand **abc** in the second line, since it is inside a quoted string that is not part of a **#define** macro definition.

Macros are not expanded while processing a **#define** or **#undef**. Thus:

```
#define abc zingo
#define xyz abc
#undef abc
xyz
```

produces **abc**. The token appearing immediately after an **#ifdef** or **#ifndef** is not expanded.

Macros are not expanded during the scan which determines the actual parameters to another macro call. Thus:

```
#define reverse(first,second)second first
#define greeting hello
reverse(greeting,
#define greeting goodbye
)
```

produces " **#define hello goodbye  hello**".

**Output**

Output consists of a copy of the input file, with modifications, plus lines of the form:

*#lineno " filename " "level"*

indicating the original source line number and filename of the following output line and whether this is the first such line after an include file has been entered (*level*=**1**), the first such line after an include file has been exited (*level*=**2**), or any other such line (*level* is empty).

**Details**

*Directory Search Order*

**#include** files is:

1. The directory of the file that contains the **#include** request (that is, **#include** is relative to the file being scanned when the request is made).
2. The directories specified by **–I** options, in left-to-right order.
3. The standard directory(s) (**/usr/include** on UNIX systems).

*Special Names*

Two special names are understood by **cpp**. The name **_ _LINE_ _** is defined as the current line number (a decimal integer) as known by **cpp**, and **_ _FILE_ _** is defined as the current filename (a C string) as known by **cpp**. They can be used anywhere (including in macros) just as any other defined name.

*Newline Characters*

A NEWLINE character terminates a character constant or quoted string. An escaped NEWLINE (that is, a backslash immediately followed by a NEWLINE) may be used in the body of a **#define** statement to continue the definition onto the next line. The escaped NEWLINE is not included in the macro value.

*Comments*

Comments are removed (unless the **–C** option is used on the command line). Comments are also ignored, except that a comment terminates a token.

**SEE ALSO**

**cc**(1V), **m4**(1V)

*Programming Utilities and Libraries*

**DIAGNOSTICS**

The error messages produced by **cpp** are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

**NOTES**

When NEWLINE characters were found in argument lists for macros to be expanded, some previous versions of **cpp** put out the NEWLINE characters as they were found and expanded. The current version of **cpp** replaces them with SPACE characters.

Because the standard directory for included files may be different in different environments, this form of **#include** directive:

**#include <file.h>**

should be used, rather than one with an absolute path, like:

**#include "/usr/include/file.h"**

**cpp** warns about the use of the absolute pathname.

While the compiler allows 8-bit strings and comments, 8-bits are not allowed anywhere else. See **cc**(1V) for an explanation about why **cc** is not 8-bit clean.

## NAME

crontab – install, edit, remove or list a user's crontab file

## SYNOPSIS

**crontab** [ *filename* ]
**crontab –e** [ *username* ]
**crontab –l** [ *username* ]
**crontab –r** [ *username* ]

## DESCRIPTION

**crontab** copies the specified file, or the standard input if no file is specified, into a directory that holds all users' **crontab** files. A user's **crontab** file lists commands that are to be executed on behalf of that user at specified times on specified dates; the format of these files is described in **crontab**(5).

If the file **/var/spool/cron/cron.allow** exists, only users whose username appears in it can use **crontab**. If that file does *not* exist, however, **crontab** checks the **/var/spool/cron/cron.deny** file to determine if the user should be denied the use of **crontab**. If neither file exists, only the super-user is allowed to submit a **crontab** job. If **cron.allow** does not exist and **cron.deny** exists and is empty, global usage is permitted. The allow/deny files consist of one user name per line.

## OPTIONS

**–e** Make a copy of the current user's **crontab** file, or create an empty file if it does not exist, and edit that file. The **vi**(1) editor will be used unless the environment variable **VISUAL** or **EDITOR** indicates an alternate editor. When editing is complete, install the file as the user's **crontab** file if it was modified. If a *username* is given, the specified user's **crontab** file is edited, rather than the current user's **crontab** file; this may only be done by the super-user.

**–l** List the user's **crontab** file.

**–r** Remove the current user's **crontab** file from the **crontab** directory. If a *username* is given, the specified user's **crontab** file is removed, rather than the current user's **crontab** file; this may only be done by the super-user.

## FILES

| | |
|---|---|
| **/var/spool/cron** | main cron directory |
| **/var/spool/cron/crontabs** | spool area |
| **/var/spool/cron/cron.allow** | list of allowed users |
| **/var/spool/cron/cron.deny** | list of denied users |

## SEE ALSO

**sh**(1), **crontab**(5), **cron**(8)

## WARNINGS

If you inadvertently enter the **crontab** command with no argument, do not attempt to get out by typing CTRL-D. This removes all entries in your **crontab** file. Instead, exit by typing your interrupt character (normally CTRL-C).

**NAME**

crypt – encode or decode a file

**SYNOPSIS**

**crypt** [ *password* ]

**DESCRIPTION**

**crypt** encrypts and decrypts the contents of a file. **crypt** reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, **crypt** demands a key from the terminal and turns off printing while the key is being typed in. **crypt** encrypts and decrypts with the same key:

**example% crypt key** < *clear.file* > *encrypted.file*
**example% crypt key** < *encrypted.file* | **pr**

will print the contents of *clear.file*.

Files encrypted by **crypt** are compatible with those treated by the editors **ed**(1), **ex**(1) and **vi**(1) in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; "sneak paths" by which keys or cleartext can become visible must be minimized.

**crypt** implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are widely known, thus **crypt** provides minimal security.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, that is, to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the **crypt** command, it is potentially visible to users executing **ps**(1) or a derivative command. To minimize this possibility, **crypt** takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of **crypt.**

**FILES**

| | |
|---|---|
| **/dev/tty** | for typed key |

**SEE ALSO**

**des**(1), **ed**(1), **ex**(1), **ps**(1), **vi**(1), **makekey**(8)

**RESTRICTIONS**

This program is not available on software shipped outside the U.S.

## NAME

csh – a shell (command interpreter) with a C-like syntax and advanced interactive features

## SYNOPSIS

**csh** [ **–bcefinstvVxX** ] [ *argument*... ]

## DESCRIPTION

**csh**, the C shell, is a command interpreter with a syntax reminiscent of C. It provides a number of convenient features for interactive use that are not available with the standard (Bourne) shell, including filename completion, command aliasing, history substitution, job control, and a number of built-in commands. As with the standard shell, the C shell provides variable, command and filename substitution.

### Initialization and Termination

When first started, the C shell normally performs commands from the **.cshrc** file in your home directory, provided that it is readable and you either own it or your real group ID matches its group ID. If the shell is invoked with a name that starts with '–', as when started by **login**(1), the shell runs as a *login* shell. In this case, after executing commands from the **.cshrc** file, the shell executes commands from the **.login** file in your home directory; the same permission checks as those for **.cshrc** are applied to this file. Typically, the **.login** file contains commands to specify the terminal type and environment.

As a login shell terminates, it performs commands from the **.logout** file in your home directory; the same permission checks as those for **.cshrc** are applied to this file.

### Interactive Operation

After startup processing is complete, an interactive C shell begins reading commands from the terminal, prompting with *hostname%* (or *hostname#* for the super-user). The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the history list and then parsed, as described under USAGE, below. Finally, the shell executes each command in the current line.

### Noninteractive Operation

When running noninteractively, the shell does not prompt for input from the terminal. A noninteractive C shell can execute a command supplied as an *argument* on its command line, or interpret commands from a script.

## OPTIONS

**–b** Force a "break" from option processing. Subsequent command-line arguments are not interpreted as C shell options. This allows the passing of options to a script without confusion. The shell does not run a set-user-ID script unless this option is present.

**–c** Read commands from the first filename *argument* (which must be present). Remaining arguments are placed in **argv**, the argument-list variable.

**–e** Exit if a command terminates abnormally or yields a nonzero exit status.

**–f** Fast start. Read neither the **.cshrc** file, nor the **.login** file (if a login shell) upon startup.

**–i** Forced interactive. Prompt for command-line input, even if the standard input does not appear to be a terminal (character-special device).

**–n** Parse (interpret), but do not execute commands. This option can be used to check C shell scripts for syntax errors.

**–s** Take commands from the standard input.

**–t** Read and execute a single command line. A '\' (backslash) can be used to escape each NEWLINE for continuation of the command line onto subsequent input lines.

**–v** Verbose. Set the **verbose** predefined variable; command input is echoed after history substitution (but before other substitutions) and before execution.

**–V** Set **verbose** before reading **.cshrc**.

**-x** Echo. Set the **echo** variable; echo commands after all substitutions and just before execution.

**-X** Set **echo** before reading **.cshrc**.

Except with the options **-c**, **-i**, **-s** or **-t**, the first nonoption *argument* is taken to be the name of a command or script. It is passed as argument zero, and subsequent arguments are added to the argument list for that command or script.

## USAGE

Refer to *SunOS User's Guide: Doing More* for tutorial information on how to use the various features of the C shell.

### Filename Completion

When enabled by setting the variable **filec**, an interactive C shell can complete a partially typed filename or user name. When an unambiguous partial filename is followed by an ESC character on the terminal input line, the shell fills in the remaining characters of a matching filename from the working directory.

If a partial filename is followed by the EOF character (usually typed as CTRL-D), the shell lists all filenames that match. It then prompts once again, supplying the incomplete command line typed in so far.

When the last (partial) word begins with a tilde (~), the shell attempts completion with a user name, rather than a file in the working directory.

The terminal bell signals errors or multiple matches; this can be inhibited by setting the variable **nobeep**. You can exclude files with certain suffixes by listing those suffixes in the variable **fignore**. If, however, the only possible completion includes a suffix in the list, it is not ignored. **fignore** does not affect the listing of filenames by the EOF character.

### Lexical Structure

The shell splits input lines into words at SPACE and TAB characters, except as noted below. The characters **&**, **|**, **;**, **<**, **>**, (, and ) form separate words; if paired, the pairs form single words. These shell metacharacters can be made part of other words, and their special meaning can be suppressed by preceding them with a '\' (backslash). A NEWLINE preceded by a \ is equivalent to a SPACE character.

In addition, a string enclosed in matched pairs of single-quotes (' ), double-quotes (" ), or backquotes (` ), forms a partial word; metacharacters in such a string, including any SPACE or TAB characters, do not form separate words. Within pairs of backquote (` ) or double-quote (" ) characters, a NEWLINE preceded by a '\' (backslash) gives a true NEWLINE character. Additional functions of each type of quote are described, below, under **Variable Substitution**, **Command Substitution**, and **Filename Substitution**.

When the shell's input is not a terminal, the character # introduces a comment that continues to the end of the input line. Its special meaning is suppressed when preceded by a \ or enclosed in matching quotes.

### Command Line Parsing

A *simple command* is composed of a sequence of words. The first word (that is not part of an I/O redirection) specifies the command to be executed. A simple command, or a set of simple commands separated by | or |& characters, forms a *pipeline*. With |, the standard output of the preceding command is redirected to the standard input of the command that follows. With |&, both the standard error and the standard output are redirected through the pipeline.

Pipelines can be separated by semicolons (;), in which case they are executed sequentially. Pipelines that are separated by && or || form conditional sequences in which the execution of pipelines on the right depends upon the success or failure, respectively, of the pipeline on the left.

A pipeline or sequence can be enclosed within parentheses '( )' to form a simple command that can be a component in a pipeline or sequence.

A sequence of pipelines can be executed asynchronously, or "in the background" by appending an '&'; rather than waiting for the sequence to finish before issuing a prompt, the shell displays the job number (see **Job Control**, below) and associated process IDs, and prompts immediately.

**History Substitution**

History substitution allows you to use words from previous command lines in the command line you are typing. This simplifies spelling corrections and the repetition of complicated commands or arguments. Command lines are saved in the history list, the size of which is controlled by the **history** variable. The most recent command is retained in any case. A history substitution begins with a **!** (although you can change this with the **histchars** variable) and may occur anywhere on the command line; history substitutions do not nest. The **!** can be escaped with \ to suppress its special meaning.

Input lines containing history substitutions are echoed on the terminal after being expanded, but before any other substitutions take place or the command gets executed.

*Event Designators*

An event designator is a reference to a command-line entry in the history list.

| | |
|---|---|
| **!** | Start a history substitution, except when followed by a SPACE character, TAB, NEWLINE, = or (. |
| **!!** | Refer to the previous command. By itself, this substitution repeats the previous command. |
| **!***n* | Refer to command-line *n*. |
| **!**–*n* | Refer to the current command-line minus *n*. |
| **!***str* | Refer to the most recent command starting with *str*. |
| **!?***str*[**?**] | Refer to the most recent command containing *str*. |
| **!{...}** | Insulate a history reference from adjacent characters (if necessary). |

*Word Designators*

A ':' (colon) separates the event specification from the word designator. It can be omitted if the word designator begins with a ^, $, *, – or %. If the word is to be selected from the previous command, the second **!** character can be omitted from the event specification. For instance, **!!:1** and **!:1** both refer to the first word of the previous command, while **!!$** and **!$** both refer to the last word in the previous command. Word designators include:

| | |
|---|---|
| **#** | The entire command line typed so far. |
| **0** | The first input word (command). |
| *n* | The *n*'th argument. |
| ^ | The first argument, that is, **1**. |
| **$** | The last argument. |
| *%* | The word matched by (the most recent) **?***s* search. |
| *x*–*y* | A range of words; –*y* abbreviates **0**–*y*. |
| * | All the arguments, or a null value if there is just one word in the event. |
| *x** | Abbreviates *x*–**$**. |
| *x*– | Like *x** but omitting word **$**. |

*Modifiers*

After the optional word designator, you can add a sequence of one or more of the following modifiers, each preceded by a **:**.

| | |
|---|---|
| **h** | Remove a trailing pathname component, leaving the head. |
| **r** | Remove a trailing suffix of the form '*.xxx*', leaving the basename. |
| **e** | Remove all but the suffix. |
| **s**/*l*/*r*[/] | Substitute *r* for *l*. |
| **t** | Remove all leading pathname components, leaving the tail. |
| **&** | Repeat the previous substitution. |
| **g** | Apply the change to the first occurrence of a match in each word, by prefixing the above (for example, **g&**). |
| **p** | Print the new command but do not execute it. |
| **q** | Quote the substituted words, escaping further substitutions. |
| **x** | Like **q**, but break into words at each SPACE character, TAB or NEWLINE. |

Unless preceded by a **g**, the modification is applied only to the first string that matches *l*; an error results if no string matches.

The left-hand side of substitutions are not regular expressions, but character strings. Any character can be used as the delimiter in place of /. A backslash quotes the delimiter character. The character &, in the right hand side, is replaced by the text from the left-hand-side. The & can be quoted with a backslash. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* from **!?***s*. You can omit the rightmost delimiter if a NEWLINE immediately follows *r*; the rightmost **?** in a context scan can similarly be omitted.

Without an event specification, a history reference refers either to the previous command, or to a previous history reference on the command line (if any).

*Quick Substitution*

^*l*^*r*[^] This is equivalent to the history substitution: **!:s**^*l*^*r*[^].

**Aliases**

The C shell maintains a list of aliases that you can create, display, and modify using the **alias** and **unalias** commands. The shell checks the first word in each command to see if it matches the name of an existing alias. If it does, the command is reprocessed with the alias definition replacing its name; the history substitution mechanism is made available as though that command were the previous input line. This allows history substitutions, escaped with a backslash in the definition, to be replaced with actual command-line arguments when the alias is used. If no history substitution is called for, the arguments remain unchanged.

Aliases can be nested. That is, an alias definition can contain the name of another alias. Nested aliases are expanded before any history substitutions is applied. This is useful in pipelines such as

**alias lm 'ls –l \!* | more'**

which when called, pipes the output of ls(1V) through **more**(1).

Except for the first word, the name of the alias may not appear in its definition, nor in any alias referred to by its definition. Such loops are detected, and cause an error message.

**I/O Redirection**

The following metacharacters indicate that the subsequent word is the name of a file to which the command's standard input, standard output, or standard error is redirected; this word is variable, command, and filename expanded separately from the rest of the command.

**<** Redirect the standard input.

**<<***word* Read the standard input, up to a line that is identical with *word*, and place the resulting lines in a temporary file. Unless *word* is escaped or quoted, variable and command substitutions are performed on these lines. Then, invoke the pipeline with the temporary file as its standard input. *word* is not subjected to variable, filename or command substitution, and each line is compared to it before any substitutions are performed by the shell.

**> >! >& >&!** Redirect the standard output to a file. If the file does not exist, it is created. If it does exist, it is overwritten; its previous contents are lost.

When set, the variable **noclobber** prevents destruction of existing files. It also prevents redirection to terminals and **/dev/null**, unless one of the **!** forms is used. The & forms redirect both standard output and the standard error (diagnostic output) to the file.

**>> >>& >>! >>&!**

Append the standard output. Like **>**, but places output at the end of the file rather than overwriting it. If **noclobber** is set, it is an error for the file not to exist, unless one of the **!** forms is used. The & forms append both the standard error and standard output to the file.

**Variable Substitution**

The C shell maintains a set of *variables*, each of which is composed of a *name* and a *value*. A variable name consists of up to 20 letters and digits, and starts with a letter (the underscore is considered a letter). A variable's value is a space-separated list of zero or more words.

To refer to a variable's value, precede its name with a '$'. Certain references (described below) can be used to select specific words from the value, or to display other information about the variable. Braces can be used to insulate the reference from other characters in an input-line word.

Variable substitution takes place after the input line is analyzed, aliases are resolved, and I/O redirections are applied. Exceptions to this are variable references in I/O redirections (substituted at the time the redirection is made), and backquoted strings (see **Command Substitution**).

Variable substitution can be suppressed by preceding the $ with a \, except within double-quotes where it always occurs. Variable substitution is suppressed inside of single-quotes. A $ is escaped if followed by a SPACE character, TAB or NEWLINE.

Variables can be created, displayed, or destroyed using the **set** and **unset** commands. Some variables are maintained or used by the shell. For instance, the **argv** variable contains an image of the shell's argument list. Of the variables used by the shell, a number are toggles; the shell does not care what their value is, only whether they are set or not.

Numerical values can be operated on as numbers (as with the **@** built-in). With numeric operations, an empty value is considered to be zero; the second and subsequent words of multiword values are ignored. For instance, when the **verbose** variable is set to any value (including an empty value), command input is echoed on the terminal.

Command and filename substitution is subsequently applied to the words that result from the variable substitution, except when suppressed by double-quotes, when **noglob** is set (suppressing filename substitution), or when the reference is quoted with the **:q** modifier. Within double-quotes, a reference is expanded to form (a portion of) a quoted string; multiword values are expanded to a string with embedded SPACE characters. When the **:q** modifier is applied to the reference, it is expanded to a list of space-separated words, each of which is quoted to prevent subsequent command or filename substitutions.

Except as noted below, it is an error to refer to a variable that is not set.

*$var*
*${var}* — These are replaced by words from the value of *var*, each separated by a SPACE character. If *var* is an environment variable, its value is returned (but ':' modifiers and the other forms given below are not available).

*$var[index]*
*${var[index]}* — These select only the indicated words from the value of *var*. Variable substitution is applied to *index*, which may consist of (or result in) a either single number, two numbers separated by a '–', or an asterisk. Words are indexed starting from 1; a '*' selects all words. If the first number of a range is omitted (as with **$argv[–2]**), it defaults to 1. If the last number of a range is omitted (as with **$argv[1–]**), it defaults to *$#var* (the word count). It is not an error for a range to be empty if the second argument is omitted (or within range).

*$#name*
*${#name}* — These give the number of words in the variable.

**$0** — This substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$*n*

${*n*} Equivalent to **$argv**[*n*].

**$*** Equivalent to **$argv**[*].

The modifiers **:e**, **:h**, **:q**, **:r**, **:t** and **:x** can be applied (see **History Substitution**), as can **:gh**, **:gt** and **:gr**. If {} (braces) are used, then the modifiers must appear within the braces. The current implementation allows only one such modifier per expansion.

The following references may not be modified with : modifiers.

$?*var*

${?*var*} Substitutes the string 1 if *var* is set or 0 if it is not set.

**$?0** Substitutes 1 if the current input filename is known, or 0 if it is not.

**$$** Substitute the process number of the (parent) shell.

**$<** Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a C shell script.

**Command and Filename Substitutions**

Command and filename substitutions are applied selectively to the arguments of built-in commands. Portions of expressions that are not evaluated are not expanded. For non-built-in commands, filename expansion of the command name is done separately from that of the argument list; expansion occurs in a subshell, after I/O redirection is performed.

**Command Substitution**

A command enclosed by backquotes (`...`) is performed by a subshell. Its standard output is broken into separate words at each SPACE character, TAB and NEWLINE; null words are discarded. This text replaces the backquoted string on the current command line. Within double-quotes, only NEWLINE characters force new words; SPACE and TAB characters are preserved. However, a final NEWLINE is ignored. It is therefore possible for a command substitution to yield a partial word.

**Filename Substitution**

Unquoted words containing any of the characters *, ?, [ or {, or that begin with ~, are expanded (also known as *globbing*) to an alphabetically sorted list of filenames, as follows:

***** Match any (zero or more) characters.

**?** Match any single character.

[...] Match any single character in the enclosed list(s) or range(s). A list is a string of characters. A range is two characters separated by a minus-sign (–), and includes all the characters in between in the ASCII collating sequence (see **ascii**(7)).

{ *str*, *str*, ... }

Expand to each string (or filename-matching pattern) in the comma-separated list. Unlike the pattern-matching expressions above, the expansion of this construct is not sorted. For instance, {b,a} expands to 'b' 'a', (not 'a' 'b'). As special cases, the characters { and }, along with the string {}, are passed undisturbed.

~[ *user* ] Your home directory, as indicated by the value of the variable **home**, or that of *user*, as indicated by the password entry for *user*.

Only the patterns *, ? and [...] imply pattern matching; an error results if no filename matches a pattern that contains them. The '.' (dot character), when it is the first character in a filename or pathname component, must be matched explicitly. The / (slash) must also be matched explicitly.

**Expressions and Operators**

A number of C shell built-in commands accept expressions, in which the operators are similar to those of C and have the same precedence. These expressions typically appear in the **@**, **exit**, **if**, **set** and **while** commands, and are often used to regulate the flow of control for executing commands. Components of an expression are separated by white space.

Null or missing values are considered 0. The result of all expressions are strings, which may represent decimal numbers.

The following C shell operators are grouped in order of precedence:

| | |
|---|---|
| ( ... ) | grouping |
| ~ | one's complement |
| ! | logical negation |
| * / % | multiplication, division, remainder (These are right associative, which can lead to unexpected results. Group combinations explicitly with parentheses.) |
| + - | addition, subtraction (also right associative) |
| << >> | bitwise shift left, bitwise shift right |
| < > <= >= | less than, greater than, less than or equal to, greater than or equal to |
| == != =~ !~ | equal to, not equal to, filename-substitution pattern match (described below), filename-substitution pattern mismatch |
| & | bitwise AND |
| ^ | bitwise XOR (exclusive or) |
| \| | bitwise inclusive OR |
| && | logical AND |
| \|\| | logical OR |

The operators: ==, !=, =~, and !~ compare their arguments as strings; other operators use numbers. The operators =~ and !~ each check whether or not a string to the left matches a filename substitution pattern on the right. This reduces the need for **switch** statements when pattern-matching between strings is all that is required.

Also available are file inquiries:

**-r** *filename* Return true, or 1 if the user has read access. Otherwise it returns false, or 0.

**-w** *filename*
True if the user has write access.

**-x** *filename*
True if the user has execute permission (or search permission on a directory).

**-e** *filename* True if *file* exists.

**-o** *filename*
True if the user owns *file*.

**-z** *filename* True if *file* is of zero length (empty).

**-f** *filename* True if *file* is a plain file.

**-d** *filename*
True if *file* is a directory.

If *file* does not exist or is inaccessible, then all inquiries return false.

An inquiry as to the success of a command is also available:

{ *command* }
If *command* runs successfully, the expression evaluates to true, 1. Otherwise it evaluates to false 0. (Note that, conversely, *command* itself typically returns 0 when it runs successfully, or some other value if it encounters a problem. If you want to get at the status directly, use the value of the **status** variable rather than this expression).

**Control Flow**

The shell contains a number of commands to regulate the flow of control in scripts, and within limits, from the terminal. These commands operate by forcing the shell either to reread input (to *loop*), or to skip input under certain conditions (to *branch*).

Each occurrence of a **foreach, switch, while, if...then** and **else** built-in must appear as the first word on its own input line.

If the shell's input is not seekable and a loop is being read, that input is buffered. The shell performs seeks within the internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward **goto** commands will succeed on nonseekable inputs.)

**Command Execution**

If the command is a C shell built-in, the shell executes it directly. Otherwise, the shell searches for a file by that name with execute access. If the command-name contains a /, the shell takes it as a pathname, and searches for it. If the command-name does not contain a /, the shell attempts to resolve it to a pathname, searching each directory in the **path** variable for the command. To speed the search, the shell uses its hash table (see the **rehash** built-in) to eliminate directories that have no applicable files. This hashing can be disabled with the **–c** or **–t**, options, or the **unhash** built-in.

As a special case, if there is no / in the name of the script and there is an alias for the word **shell**, the expansion of the **shell** alias is prepended (without modification), to the command line. The system attempts to execute the first word of this special (late-occurring) alias, which should be a full pathname. Remaining words of the alias's definition, along with the text of the input line, are treated as arguments.

When a pathname is found that has proper execute permissions, the shell forks a new process and passes it, along with its arguments to the kernel (using the **execve**(2V) system call). The kernel then attempts to overlay the new process with the desired program. If the file is an executable binary (in **a.out**(5) format) the kernel succeeds, and begins executing the new process. If the file is a text file, and the first line begins with **#!**, the next word is taken to be the pathname of a shell (or command) to interpret that script. Subsequent words on the first line are taken as options for that shell. The kernel invokes (overlays) the indicated shell, using the name of the script as an argument.

If neither of the above conditions holds, the kernel cannot overlay the file (the **execve**(2V) call fails); the C shell then attempts to execute the file by spawning a new shell, as follows:

- If the first character of the file is a **#**, a C shell is invoked.
- Otherwise, a standard (Bourne) shell is invoked.

**Signal Handling**

The shell normally ignores QUIT signals. Background jobs are immune to signals generated from the keyboard, including hangups (HUP). Other signals have the values that the C shell inherited from its environment. The shell's handling of interrupt and terminate signals within scripts can be controlled by the **onintr** built-in. Login shells catch the TERM signal; otherwise this signal is passed on to child processes. In no case are interrupts allowed when a login shell is reading the **.logout** file.

**Job Control**

The shell associates a numbered *job* with each command sequence, to keep track of those commands that are running in the background or have been stopped with TSTP signals (typically CTRL-Z). When a command, or command sequence (semicolon separated list), is started in the background using the **&** metacharacter, the shell displays a line with the job number in brackets, and a list of associated process numbers:

**[1] 1234**

To see the current list of jobs, use the **jobs** built-in command. The job most recently stopped (or put into the background if none are stopped) is referred to as the *current* job, and is indicated with a '+'. The previous job is indicated with a '–'; when the current job is terminated or moved to the foreground, this job takes its place (becomes the new current job).

To manipulate jobs, refer to the **bg**, **fg**, **kill**, **stop** and **%** built-ins.

A reference to a job begins with a '**%**'. By itself, the percent-sign refers to the current job.

| | |
|---|---|
| **%** **%+** **%%** | The current job. |
| **%–** | The previous job. |
| *%j* | Refer to job *j* as in: '**kill –9** *%j*'. *j* can be a job number, or a string that uniquely specifies the command-line by which it was started; '**fg %vi**' might bring a stopped **vi** job to the foreground, for instance. |

*%?string* Specify the job for which the command-line uniquely contains *string*.

A job running in the background stops when it attempts to read from the terminal. Background jobs can normally produce output, but this can be suppressed using the '**stty tostop**' command.

**Status Reporting**

While running interactively, the shell tracks the status of each job and reports whenever a finishes or becomes blocked. It normally displays a message to this effect as it issues a prompt, so as to avoid disturbing the appearance of your input. When set, the **notify** variable indicates that the shell is to report status changes immediately. By default, the **notify** command marks the current process; after starting a background job, type **notify** to mark it.

**Built-In Commands**

Built-in commands are executed within the C shell. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell.

**:** Null command. This command is interpreted, but performs no action.

**alias** [ *name* [ *def* ] ]

Assign *def* to the alias *name*. *def* is a list of words that may contain escaped history-substitution metasyntax. *name* is not allowed to be **alias** or **unalias**. If *def* is omitted, the alias *name* is displayed along with its current definition. If both *name* and *def* are omitted, all aliases are displayed.

**bg** [*%job*] ...

Run the current or specified jobs in the background.

**break** Resume execution after the **end** of the nearest enclosing **foreach** or **while** loop. The remaining commands on the current line are executed. This allows multilevel breaks to be written as a list of **break** commands, all on one line.

**breaksw** Break from a **switch**, resuming after the **endsw**.

**case** *label*: A label in a **switch** statement.

**cd** [ *dir* ]
**chdir** [ *dir* ]

Change the shell's working directory to directory *dir*. If no argument is given, change to the home directory of the user. If *dir* is a relative pathname not found in the current directory, check for it in those directories listed in the **cdpath** variable. If *dir* is the name of a shell variable whose value starts with a /, change to the directory named by that value.

**continue** Continue execution of the nearest enclosing **while** or **foreach**.

**default:** Labels the default case in a **switch** statement. The default should come after all **case** labels. Any remaining commands on the command line are first executed.

**dirs** [ **–l** ]

Print the directory stack, most recent to the left; the first directory shown is the current directory. With the **–l** argument, produce an unabbreviated printout; use of the ~ notation is suppressed.

**echo** [ **–n** ] *list*

The words in *list* are written to the shell's standard output, separated by SPACE characters. The output is terminated with a NEWLINE unless the **–n** option is used.

**eval** *argument* ...

Reads the arguments as input to the shell, and executes the resulting command(s). This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See **tset**(1) for an example of how to use **eval**.

**exec** *command*

Execute *command* in place of the current shell, which terminates.

**exit** [ (*expr*) ]

The shell exits, either with the value of the **status** variable, or with the value of the specified by the expression *expr*.

**fg** % [ *job* ]

Bring the current or specified *job* into the foreground.

**foreach** *var* (*wordlist*)

...

**end**

The variable *var* is successively set to each member of *wordlist*. The sequence of commands between this command and the matching **end** is executed for each new value of *var*. (Both **foreach** and **end** must appear alone on separate lines.)

The built-in command **continue** may be used to continue the loop prematurely and the built-in command **break** to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with ? before any statements in the loop are executed.

**glob** *wordlist*

Perform filename expansion on *wordlist*. Like **echo**, but no \ escapes are recognized. Words are delimited by null characters in the output.

**goto** *label*

The specified *label* is filename and command expanded to yield a label. The shell rewinds its input as much as possible and searches for a line of the form *label*: possibly preceded by SPACE or TAB characters. Execution continues after the indicated line. It is an error to jump to a label that occurs between a **while** or **for** built-in, and its corresponding **end**.

**hashstat**

Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding **execs**). An **exec** is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component that does not begin with a '/'.

**history** [ **–hr** ] [ *n* ]

Display the history list; if *n* is given, display only the *n* most recent events.

**–r** Reverse the order of printout to be most recent first rather than oldest first.

**–h** Display the history list without leading numbers. This is used to produce files suitable for sourcing using the **–h** option to *source*.

**if** (*expr*) *command*

If the specified expression evaluates to true, the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Note: I/O redirection occurs even if *expr* is false, when *command* is *not* executed (this is a bug).

**if** (*expr*) **then**

...

**else if** (*expr2*) **then**

...

**else**

...

**endif**

If *expr* is true, commands up to the first **else** are executed. Otherwise, if *expr2* is true, the commands between the **else if** and the second **else** are executed. Otherwise, commands between the **else** and the **endif** are executed. Any number of **else if** pairs are allowed, but only one **else**. Only one **endif** is needed, but it is required. The words **else** and **endif** must be the first nonwhite characters on a line. The **if** must appear alone on its input line or after an **else**.)

**jobs**[ **–l** ]

List the active jobs under job control.

**–l** List process IDs, in addition to the normal information.

**kill** [ *–sig* ] [ *pid* ] [ *%job* ] ...

**kill –l** Send the TERM (terminate) signal, by default, or the signal specified, to the specified process ID, the *job* indicated, or the current *job*. Signals are either given by number or by name. There is no default. Typing **kill** does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process is sent a CONT (continue) signal as well.

**–l** List the signal names that can be sent.

**limit** [ **–h** ] [ *resource* [ *max-use* ] ]

Limit the consumption by the current process or any process it spawns, each not to exceed *max-use* on the specified *resource*. If *max-use* is omitted, print the current limit; if *resource* is omitted, display all limits.

**–h** Use hard limits instead of the current limits. Hard limits impose a ceiling on the values of the current limits. Only the super-user may raise the hard limits.

*resource* is one of:

| | |
|---|---|
| **cputime** | Maximum CPU seconds per process. |
| **filesize** | Largest single file allowed. |
| **datasize** | Maximum data size (including stack) for the process. |
| **stacksize** | Maximum stack size for the process. |
| **coredumpsize** | Maximum size of a core dump (file). |
| **descriptors** | Maximum value for a file descriptor. |

*max-use* is a number, with an optional scaling factor, as follows:

| | |
|---|---|
| *n***h** | Hours (for **cputime**). |
| *n***k** | *n* kilobytes. This is the default for all but **cputime**. |
| *n***m** | *n* megabytes or minutes (for **cputime**). |
| *mm*:*ss* | Minutes and seconds (for **cputime**). |

**login** [ *username* |–p ]

Terminate a login shell and invoke **login**(1). The **.logout** file is not processed. If *username* is omitted, *login* prompts for the name of a user.

**–p** Preserve the current environment (variables).

**logout** Terminate a login shell.

**nice** [ *+n* |*–n* ] [ *command* ]

Increment the process priority value for the shell or for *command* by *n*. The higher the priority value, the lower the priority of a process, and the slower it runs. When given, *command* is always run in a subshell, and the restrictions placed on commands in simple **if** commands apply. If *command* is omitted, **nice** increments the value for the current shell. If no increment is specified, **nice** sets the nice value to 4. The range of nice values is from –20 through 19. Values of *n* outside this range set the value to the lower, or to the higher boundary, respectively.

*+n* Increment the process priority value by *n*.
*–n* Decrement by *n*. This argument can be used only by the super-user.

**nohup** [ *command* ]

Run *command* with HUPs ignored. With no arguments, ignore HUPs throughout the remainder of a script. When given, *command* is always run in a subshell, and the restrictions placed on commands in simple **if** commands apply. All processes detached with **&** are effectively **nohup**'d.

**notify** [ *%job* ] ...

Notify the user asynchronously when the status of the current, or of specified jobs, changes.

**onintr** [ – | *label*]

Control the action of the shell on interrupts. With no arguments, **onintr** restores the default action of the shell on interrupts. (The shell terminates shell scripts and returns to the terminal command input level). With the – argument, the shell ignores all interrupts. With a *label* argument, the shell executes a **goto** *label* when an interrupt is received or a child process terminates because it was interrupted.

**popd** [+*n*]

Pop the directory stack, and **cd**s to the new top directory. The elements of the directory stack are numbered from 0 starting at the top.

+*n* Discard the *n*'th entry in the stack.

**pushd** [+*n* | *dir*]

Push a directory onto the directory stack. With no arguments, exchange the top two elements.

+*n* Rotate the *n*'th entry to the top of the stack and **cd** to it.
*dir* Push the current working directory onto the stack and change to *dir*.

**rehash** Recompute the internal hash table of the contents of directories listed in the *path* variable to account for new commands added.

**repeat** *count command*

Repeat *command count* times *command* is subject to the same restrictions as with the one-line **if** statement.

**set** [*var* [ = *value* ] ]
**set** *var*[*n*] = *word*

With no arguments, **set** displays the values of all shell variables. Multiword values are displayed as a parenthesized list. With the *var* argument alone, **set** assigns an empty (null) value to the variable *var*. With arguments of the form *var* = *value* **set** assigns *value* to *var*, where *value* is one of:

*word* A single word (or quoted string).
(*wordlist*) A space-separated list of words enclosed in parentheses.

Values are command and filename expanded before being assigned. The form **set** *var*[*n*] = *word* replaces the *n*'th word in a multiword value with *word*.

**setenv** [ *VAR* [ *word* ] ]

With no arguments, **setenv** displays all environment variables. With the *VAR* argument sets the environment variable *VAR* to have an empty (null) value. (By convention, environment variables are normally given upper-case names.) With both *VAR* and *word* arguments **setenv** sets the environment variable NAME to the value *word*, which must be either a single word or a quoted string. The most commonly used environment variables, **USER**, **TERM**, and **PATH**, are automatically imported to and exported from the **csh** variables **user**, **term**, and **path**; there is no need to use **setenv** for these. In addition, the shell sets the **PWD** environment variable from the **csh** variable **cwd** whenever the latter changes.

**shift** [ *variable* ]

The components of **argv**, or *variable*, if supplied, are shifted to the left, discarding the first component. It is an error for the variable not to be set, or to have a null value.

**source** [ **–h** ] *name*

Reads commands from *name*. **source** commands may be nested, but if they are nested too deeply the shell may run out of file descriptors. An error in a sourced file at any level terminates all nested **source** commands.

**–h** Place commands from the file *name* on the history list without executing them.

**stop** [*%job*] ...
Stop the current or specified background job.

**suspend** Stop the shell in its tracks, much as if it had been sent a stop signal with ^Z. This is most often used to stop shells started by **su**.

**switch** (*string*)
**case** *label*:
...
**breaksw**
...
**default:**
...
**breaksw**
**endsw** Each *label* is successively matched, against the specified *string*, which is first command and filename expanded. The file metacharacters *, ? and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a "default" label is found, execution begins after the default label. Each **case** statement and the **default** statement must appear at the beginning of a line. The command **breaksw** continues execution after the **endsw**. Otherwise control falls through subsequent **case** and **default** statements as with C. If no label matches and there is no default, execution continues after the **endsw**.

**time** [ *command* ]
With no argument, print a summary of time used by this C shell and its children. With an optional *command*, execute *command* and print a summary of the time it uses.

**umask** [ *value* ]
Display the file creation mask. With *value* set the file creation mask. *value* is given in octal, and is XORed with the permissions of 666 for files and 777 for directories to arrive at the permissions for new files. Common values include 002, giving complete access to the group, and read (and directory search) access to others, or 022, giving read (and directory search) but not write permission to the group and others.

**unalias** *pattern*
Discard aliases that match (filename substitution) *pattern*. All aliases are removed by **unalias** *.

**unhash** Disable the internal hash table.

**unlimit** [ **–h** ] [ *resource* ]
Remove a limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed. See the description of the **limit** command for the list of *resource* names.

**–h** Remove corresponding hard limits. Only the super-user may do this.

**unset** *pattern*
Remove variables whose names match (filename substitution) *pattern*. All variables are removed by '**unset** *'; this has noticeably distasteful side-effects.

**unsetenv** *variable*
Remove *variable* from the environment. Pattern matching, as with **unset** is not performed.

**wait** Wait for background jobs to finish (or for an interrupt) before prompting.

**while** (*expr*)
...
**end** While *expr* is true (evaluates to non-zero), repeat commands between the **while** and the matching **end** statement. **break** and **continue** may be used to terminate or continue the loop prematurely. The **while** and **end** must appear alone on their input lines. If the shell's input is a terminal, it prompts for commands with a question-mark until the **end** command is entered and then performs the commands in the loop.

**%**[ *job* ] [ **&** ]

Bring the current or indicated *job* to the foreground. With the ampersand, continue running *job* in the background.

**@** [ *var* =*expr* ]
**@** [ *var*[*n*] =*expr* ]

With no arguments, display the values for all shell variables. With arguments, the variable *var*, or the *n*'th word in the value of *var*, to the value that *expr* evaluates to. (If [*n*] is supplied, both *var* and its *n*'th component must already exist.)

If the expression contains the characters **>**, **<**, **&** or **|**, then at least this part of *expr* must be placed within parentheses.

The operators ***=**, **+=**, etc., are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* that would otherwise be single words.

Special postfix operators, **+ +** and **– –** increment or decrement *name*, respectively.

**Environment Variables and Predefined Shell Variables**

Unlike the standard shell, the C shell maintains a distinction between environment variables, which are automatically exported to processes it invokes, and shell variables, which are not. Both types of variables are treated similarly under variable substitution. The shell sets the variables **argv**, **cwd**, **home**, **path**, **prompt**, **shell**, and **status** upon initialization. The shell copies the environment variable **USER** into the shell variable **user**, **TERM** into **term**, and **HOME** into **home**, and copies each back into the respective environment variable whenever the shell variables are reset. **PATH** and **path** are similarly handled. You need only set **path** once in the **.cshrc** or **.login** file. The environment variable **PWD** is set from **cwd** whenever the latter changes. The following shell variables have predefined meanings:

**argv** Argument list. Contains the list of command line arguments supplied to the current invocation of the shell. This variable determines the value of the positional parameters **$1**, **$2**, and so on.

**cdpath** Contains a list of directories to be searched by the **cd**, **chdir**, and **popd** commands, if the directory argument each accepts is not a subdirectory of the current directory.

**cwd** The full pathname of the current directory.

**echo** Echo commands (after substitutions), just before execution.

**fignore** A list of filename suffixes to ignore when attempting filename completion. Typically the single word '**.o**'.

**filec** Enable filename completion, in which case the CTRL-D character CTRL-D) and the ESC character have special significance when typed in at the end of a terminal input line:

EOT Print a list of all filenames that start with the preceding string.
ESC Replace the preceding string with the longest unambiguous extension.

**hardpaths** If set, pathnames in the directory stack are resolved to contain no symbolic-link components.

**histchars** A two-character string. The first character replaces **!** as the history-substitution character. The second replaces the carat (**^**) for quick substitutions.

**history** The number of lines saved in the history list. A very large number may use up all of the C shell's memory. If not set, the C shell saves only the most recent command.

**home** The user's home directory. The filename expansion of **~** refers to the value of this variable.

**ignoreeof** If set, the shell ignores EOF from terminals. This protects against accidentally killing a C shell by typing a CTRL-D.

**mail** A list of files where the C shell checks for mail. If the first word of the value is a number, it specifies a mail checking interval in seconds (default 5 minutes).

**nobeep** Suppress the bell during command completion when asking the C shell to extend an ambiguous filename.

**noclobber** Restrict output redirection so that existing files are not destroyed by accident. **>** redirections can only be made to new files. **>>** redirections can only be made to existing files.

**noglob** Inhibit filename substitution. This is most useful in shell scripts once filenames (if any) are obtained and no further expansion is desired.

**nonomatch** Returns the filename substitution pattern, rather than an error, if the pattern is not matched. Malformed patterns still result in errors.

**notify** If set, the shell notifies you immediately as jobs are completed, rather than waiting until just before issuing a prompt.

**path** The list of directories in which to search for commands. **path** is initialized from the environment variable **PATH**, which the C shell updates whenever **path** changes. A null word specifies the current directory. The default is typically: **(. /usr/ucb /usr/bin)**. If **path** becomes unset only full pathnames will execute. An interactive C shell will normally hash the contents of the directories listed after reading **.cshrc**, and whenever **path** is reset. If new commands are added, use the **rehash** command to update the table.

**prompt** The string an interactive C shell prompts with. Noninteractive shells leave the **prompt** variable unset. Aliases and other commands in the **.cshrc** file that are only useful interactively, can be placed after the following test: **'if ($?prompt == 0) exit'**, to reduce startup time for noninteractive shells. A **!** in the **prompt** string is replaced by the current event number. The default prompt is *hostname***%** for mere mortals, or *hostname*# for the super-user.

**savehist** The number of lines from the history list that are saved in ~**/.history** when the user logs out. Large values for **savehist** slow down the C shell during startup.

**shell** The file in which the C shell resides. This is used in forking shells to interpret files that have execute bits set, but that are not executable by the system.

**status** The status returned by the most recent command. If that command terminated abnormally, 0200 is added to the status. Built-in commands that fail return exit status 1, all other built-in commands set status to 0.

**time** Control automatic timing of commands. Can be supplied with one or two values. The first is the reporting threshold in CPU seconds. The second is a string of tags and text indicating which resources to report on. A tag is a percent sign (**%**) followed by a single *upper-case* letter (unrecognized tags print as text):

| | |
|---|---|
| **%D** | Average amount of unshared data space used in Kilobytes. |
| **%E** | Elapsed (wallclock) time for the command. |
| **%F** | Page faults. |
| **%I** | Number of block input operations. |
| **%K** | Average amount of unshared stack space used in Kilobytes. |
| **%M** | Maximum real memory used during execution of the process. |
| **%O** | Number of block output operations. |
| **%P** | Total CPU time — U (user) plus S (system) — as a percentage of E (elapsed) time. |

**%S** Number of seconds of CPU time consumed by the kernel on behalf of the user's process.
**%U** Number of seconds of CPU time devoted to the user's process.
**%W** Number of swaps.
**%X** Average amount of shared memory used in Kilobytes.

The default summary display outputs from the **%U**, **%S**, **%E**, **%P**, **%X**, **%D**, **%I**, **%O**, **%F** and **%W** tags, in that order.

**verbose** Display each command after history substitution takes place.

## ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout all command line parsing. These variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## FILES

| | |
|---|---|
| **~/.cshrc** | Read at beginning of execution by each shell. |
| **~/.login** | Read by login shells after **.cshrc** at login. |
| **~/.logout** | Read by login shells at logout. |
| **~/.history** | Saved history for use at next login. |
| **/tmp/sh*** | Temporary file for '**<<**'. |
| **/etc/passwd** | Source of home directories for '*~name*'. |

## SEE ALSO

**login**(1), **printenv**(1), **sh**(1), **tset**(1), **access**(2V), **execve**(2V), **fork**(2V), **pipe**(2V), **termio**(4), **a.out**(5), **environ**(5V), **locale**(5), **ascii**(7), **iso_8859_1**(7)

*SunOS User's Guide: Doing More*
*SunOS User's Guide: Getting Started*

## DIAGNOSTICS

**You have stopped jobs.**

You attempted to exit the C shell with stopped jobs under job control. An immediate second attempt to exit will succeed, terminating the stopped jobs.

## LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 1,048,576 characters. However, the maximum number of arguments to a command for which filename expansion applies is 1706. Command substitutions may expand to no more characters than are allowed in the argument list. To detect looping, the shell restricts the number of **alias** substitutions on a single line to 20.

## BUGS

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory; this can be misleading (that is, wrong) as the job may have changed directories internally.

Shell built-in functions are not stoppable/restartable. Command sequences of the form *a* ; *b* ; *c* are also not handled gracefully when stopping is attempted. If you suspend *b*, the shell never executes *c*. This is especially noticeable if the expansion results from an alias. It can be avoided by placing the sequence in parentheses to force it into a subshell.

Control over terminal output after processes are started is primitive; use the Sun Window system if you need better output control.

Multiline shell procedures should be provided, as they are with the standard (Bourne) shell.

Commands within loops, prompted for by ?, are not placed in the *history* list.

Control structures should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. There are two problems with : modifier usage on variable substitutions: not all of the modifiers are available, and only one modifier per substitution is allowed.

The g (global) flag in history substitutions applies only to the first match in each word, rather than all matches in all words. The standard text editors consistently do the latter when given the g flag in a substitution command.

Quoting conventions are confusing. Overriding the escape character to force variable substitutions within double quotes is counterintuitive and inconsistent with the Bourne shell.

Symbolic links can fool the shell. Setting the **hardpaths** variable alleviates this.

'**set path**' should remove duplicate pathnames from the pathname list. These often occur because a shell script or a **.cshrc** file does something like '**set path=(/usr/local /usr/hosts $path)**' to ensure that the named directories are in the pathname list.

The only way to direct the standard output and standard error separately is by invoking a subshell, as follows:

**example%** (*command* > *outfile*) >& *errorfile*

Although robust enough for general use, adventures into the esoteric periphery of the C shell may reveal unexpected quirks.

## NAME

csplit – split a file with respect to a given context

## SYNOPSIS

**csplit** [ **–f** *prefix* ] [ **–k** ] [ **–s** ] *filename argument1* [ *...argumentn* ]

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**csplit** reads the file whose name is *filename* and separates it into *n*+1 sections, defined by the arguments *argument1* through *argumentn*. If the *filename* argument is a '–', the standard input is used. By default the sections are placed in files named **xx00** through **xx***n*. *n* may not be greater than 99. These sections receive the following portions of the file:

**xx00** From the start of *filename* up to (but not including) the line indicated by *argument1* (see OPTIONS below for an explanation of these arguments.)

**xx01:** From the line indicated by *argument1* up to the line indicated by *argument2*.

**xx***n***:** From the line referenced by *argumentn* to the end of *filename*.

**csplit** prints the character counts for each file created, and removes any files it creates if an error occurs.

## OPTIONS

**–f** *prefix* name the created files *prefix***00** through *prefix**n*.

**–k** Suppress removal of created files when an error occurs.

**–s** Suppress printing of character counts.

The arguments *argument1* through *argumentn* can be a combination of the following selection operators:

*/rexp/* A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line then becomes the line containing *rexp*. This argument may be followed by an optional '+' or '–' some number of lines (for example, **/Page/–5**).

*%rexp%* This argument is the same as */rexp/*, except that no file is created for the selected section.

*lineno* A file is to be created from the current line up to (but not including) *lineno*. The current line becomes *lineno*.

{*num*} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lineno*, the file will be split every *lineno* lines (*num* times) from that point.

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotes. Regular expressions may not contain embedded new-lines.

## EXAMPLES

This example splits the file at every 100 lines, up to 10,000 lines.

```
csplit –k file 100 {99}
```

Assuming that **prog.c** follows the normal C coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in **prog.c**.

```
csplit –k prog.c '%main(%' '/^}/+1' {20}
```

## SEE ALSO

**ed**(1), **sh**(1), **regexp**(3)

**DIAGNOSTICS**

Self-explanatory except for:

**arg – out of range**

which means that the given argument did not refer to a line between the current position and the EOF.

## NAME

ctags – create a tags file for use with ex and vi

## SYNOPSIS

**ctags** [ **–aBFtuvwx** ] [ **–f** *tagsfile* ] *filename*...

## DESCRIPTION

**ctags** makes a tags file for **ex**(1) from the specified C, Pascal, FORTRAN, **yacc**(1), and **lex**(1) sources. A tags file gives the locations of specified objects (in this case functions and type definitions) in a group of files. Each entry in the tags is composed of three fields separated by white space, the object name, the file in which it is defined, and an address specification. Function definitions are located using regular expression patterns, type definitions, using a line number.

**ex** and **vi**(1) use entries in the tags file to locate and display a definition.

Normally **ctags** places the tag descriptions in a file called **tags**; this may be overridden with the **–f** option. By default, the tags file is sorted in lexicographic (ASCII) order, and **ex** expects its entries to be so sorted.

Files with names ending in **.c** or **.h** are assumed to be C source files and are searched for C routine and macro definitions. Files with names ending in **.y** are assumed to be **yacc** source files. Files with names ending in **.l** are assumed to be **lex** files. Others are first examined to see if they contain any Pascal or FORTRAN routine definitions; if not, they are processed again looking for C definitions.

The tag for the **main**( ) function is treated specially in C programs. The tag formed is created by prepending **M** to *filename*, with a trailing **.c** removed, if any, and leading pathname components also removed. This makes use of **ctags** practical in directories with more than one program.

## OPTIONS

**–a** Append output to an existing tags file. The resulting file is not sorted. To preserve the order, use **–u** instead.

**–B** Use backward searching patterns (?...?).

**–F** Use forward searching patterns (/.../) (default).

**–t** Create tags for typedefs.

**–u** Update the specified files in the tags file. Entries that refer to them are deleted and then replaced in lexicographic order. Beware: this option is implemented in a way which is rather slow; it may be faster simply to rebuild the tags file.

**–v** Produce an index of the form expected by **vgrind**(1) on the standard output. This listing contains the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through '**sort –f**'. See EXAMPLES.

**–w** Suppress warning diagnostics.

**–x** Produce a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

## EXAMPLES

Using **ctags** with the **–v** option produces entries in an order which may not always be appropriate for **vgrind**. To produce results in alphabetical order, you may want to run the output through '**sort –f**'.

```
example% ctags –v filename.c filename.h | sort –f > index
example% vgrind –x index
```

To build a tags file for C sources in a directory hierarchy, first create an empty tags file, and then run the following **find**(1) command:

```
example% cd ~/src ; rm -f tags ; touch tags
example% find ~/src \( -name '*.c' -o -name '*.h' \) -exec ctags -u -f /usr/src/tags {} \;
```

**FILES**

| | |
|---|---|
| **tags** | output tags file |

**SEE ALSO**

**cc**(1V), **ex**(1), **find**(1), **vgrind**(1), **vi**(1)

**NOTES**

While the compiler allows 8-bit strings and comments, 8-bits are not allowed anywhere else. See **cc**(1V) for an explanation about why **cc** is not 8-bit clean.

**BUGS**

Recognition of **functions**, **subroutines** and **procedures** for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name, **ctags** will only make an entry for one.

**ctags** does not know about **#ifdefs**.

**ctags** should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of **–tx** shows only the last line of typedefs.

**NAME**

ctrace – generate a C program execution trace

**SYNOPSIS**

**ctrace** [ **–f** *functions* ] [ **–v** *functions* ] [ **–o x u e** ] [ **–s P b** ] [ **–l** *n* ] [ **–t** *n* ] [ ] [ **–b** ] [ **–p** *'s'* ] [ **–r** *f* ] [ *filename* ]

**AVAILABILITY**

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**ctrace** allows you to follow the execution of a C program, statement by statement. The effect is similar to executing a shell procedure with the **–x** option. **ctrace** reads the C program in *filename* (or from standard input if you do not specify *filename*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of **ctrace** into a temporary file because the **cc**(1V) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed.

A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the **tail**(1) command.

**OPTIONS**

**–f** *functions* Trace only these *functions*.

**–v** *functions* Trace all but these *functions*.

You may want to add to the default formats for printing variables. **long** and pointer variables are always printed as signed integers. Pointers to character arrays are printed as strings if appropriate. **char**, **short**, and **int** variables are printed as signed integers and, if appropriate, as characters. **double** variables are printed as floating-point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these options:

**–o** Octal.

**–x** Hexadecimal.

**–u** Unsigned.

**–e** Floating point.

These options are used only in special circumstances:

**–l** *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.

**–s** Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.

**–t** *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The DIAGNOSTICS section explains when to use this option.

**–P** Run the C preprocessor on the input before tracing it. You can also use the **–D**, **–I**, and **–U** **cc**(1V) options.

These options are used to tailor the run-time trace package when the traced program will run in a non-UNIX system environment:

**-b** Use only basic functions in the trace code, that is, those in **ctype**(3V), **printf**(3V), and **string**(3). These are often available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have **signal**(3V), **fflush**, **longjmp** or **setjmp**(3V) (see **fclose**(3V) and **setjmp**(3V)).

**-p** *'s'* Change the trace print function from the default of **'printf('**. For example, **'fprintf(stderr'** would send the trace to the standard error output.

**-r***f* Use file *f* in place of the **runtime.c** trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **-p** option).

## USAGE

### Execution-Time Trace Control

The default operation for **ctrace** is to trace the entire program file, unless you use the **-f** or **-v** options to trace specific functions. This does not give you statement by statement control of the tracing, nor does it let you turn the tracing off and on when executing the tracedprogram.

You can do both of these by adding **ctroff()** and **ctron()** function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with **if** statements, and you can even conditionally include this code because **ctrace** defines the CTRACE preprocessor variable. For example:

```
#ifdef
CTRACE
        if (c == '!' && i > 1000)
                ctron();
#endif
```

You can also call these functions from **dbx**(1) if you compile with the **-g** option. For example, to trace all but lines 7 to 10 in the primary source file, enter:

```
dbx a.out
when at 7 { call ctroff(); cont; }
when at 11 { call ctron(); cont; }
run
```

You can also turn the trace off and on by setting the static variable **tr_ct_** to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly, such as **adb**(1).

## EXAMPLES

If the file **lc.c** contains this C program:

```
#include <stdio.h>
main()  /* count lines in input */
{
        int c, nl;
        nl = 0;
        while ((c = getchar()) != EOF)
                if (c = '\n')
                        ++nl;
        printf("%d\n", nl);
}
```

and you enter these commands and test data:

```
cc lc.c
a.out
1
CTRL-D,
```

the program will be compiled and executed. The output of the program will be the number **2**, which is not correct because there is only one line in the test data. The error in this program is common, but subtle.

If you invoke **ctrace** with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
main()
        nl = 0;
        /* nl == 0 */
        while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```
        /* c == 49 or '1' */
                if (c = '\n')
                /* c == 10 or '\n' */
                        ++nl;
                        /* nl == 1 */
        while ((c = getchar()) != EOF)
        /* c == 10 or '\n' */
                if (c = '\n')
                /* c == 10 or '\n' */
                        ++nl;
                        /* nl == 2 */
/* repeating */
```

If you now enter an end of file character (CTRL-D) the final output will be:

```
/* repeated <1 time */
        while ((c = getchar()) != EOF)
        /* c == -1 */
        printf("%d\n", nl);
        /* nl == 2 */2
/* return */
```

Program output is printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by **ctrace** at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator = in place of the equal operator ==. You can easily miss this error during code reading.

## FILES

**/usr/lib/ctrace/runtime.c** run-time trace package

## SEE ALSO

**adb**(1), **cc**(1V), **dbx**(1), **tail**(1), **ctype**(3V), **fclose**(3V), **printf**(3V), **setjmp**(3V), **signal**(3V), **string**(3)

## DIAGNOSTICS

This section contains diagnostic messages from both **ctrace** and **cc**(1V), since the traced code often gets some **cc** warning messages. You can get **cc** error messages in some rare cases, all of which can be avoided.

### From ctrace

**warning: some variables are not traced in this statement**

Only 10 variables are traced in a statement to prevent the C compiler '**out of tree space; simplify expression**' error. Use the **–t** option to increase this number.

**warning: statement too long to trace**

This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

**cannot handle preprocessor code, use –P option**
This is usually caused by **#ifdef/#endif** preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

**'if ... else if' sequence too long**
Split the sequence by removing an **else** from the middle.

**possible syntax error, try –P option**
Use the **–P** option to preprocess the **ctrace** input, along with any appropriate **–D**, **–I**, and **–U** preprocessor options. If you still get the error message, check the WARNINGS section below.

**From cc**

**warning: floating-point not implemented**
**warning: illegal combination of pointer and integer**
**warning: statement not reached**
**warning: sizeof returns 0**
Ignore these messages.

**compiler takes size of function**
See the **ctrace 'possible syntax error'** message above.

**yacc stack overflow**
See the **ctrace ''if ... else if' sequence too long'** message above.

**out of tree space; simplify expression**
Use the **–t** option to reduce the number of traced variables per statement from the default of 10. Ignore the **'ctrace: too many variables to trace'** warnings you will now get.

**redeclaration of signal**
Either correct this declaration of **signal**(3V), or remove it and **#include <signal.h>**.

**Warnings**

You will get a **ctrace** syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

**ctrace** assumes that BADMAG is a preprocessor macro, and that EOF and NULL are **#defined** constants. Declaring any of these to be variables, for example, **'int EOF;'**, will cause a syntax error.

## BUGS

**ctrace** does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. **ctrace** may choose to print the address of an aggregate or use the wrong format (for example, **%e** for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

## NAME

cu – connect to remote system

## SYNOPSIS

**cu** [ **–dhnt** ] [ **–e** | **–o** ] [ **–l** *line* ] [ **–s** *speed* ] *phone-number*

**cu** [ **–dh** ] [ **–e** | **–o** ] [ **–s** *speed* ] **–l** *line*

**cu** [ **–dh** ] [ **–e** | **–o** ] *systemname*

## DESCRIPTION

**cu** calls up another system, or possibly a terminal. It manages an interactive conversation with possible transfers of ASCII files.

If *phone-number* is specified, it is the telephone number of the system to be dialed. Equal signs specify a pause for a secondary dial tone, and minus signs specify dialing delays of 4 seconds. If *systemname* is specified, it is the UUCP name of a the system to be dialed; in this case, **cu** will obtain an appropriate direct line or telephone number from **/etc/uucp/Systems**. Note: the *systemname* option should not be used in conjunction with the **–l** and **–s** options as **cu** will connect to the first available line for the system name specified, ignoring the requested line and speed. If neither *phone-number* nor *systemname* are specified, the **–l** *line* option must be provided; *line* specifies the device name to use.

## OPTIONS

**–d** Print diagnostic traces.

**–h** Emulate local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.

**–n** For added security, prompt the user to provide the telephone number to be dialed rather than taking it from the command line.

**–t** Perform appropriate mapping of RETURN to RETURN-LINE-FEED pairs. This is used to dial an ASCII terminal which has been set to auto answer.

**–e** Generate even parity for data sent to the remote system.

**–o** Generate odd parity for data sent to the remote system.

**–l** *line* Specify a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line having the right speed. When the **–l** option is used without the **–s** option, the speed of a line is taken from the **/etc/uucp/Devices** file. When the **–l** and **–s** options are both used together, **cu** will search the **Devices** file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made. The specified device is generally a directly connected asynchronous line (for example, **/dev/tty***ab*) in which case a telephone number (*phone-number*) is not required. The specified device need not be in the **/dev** directory. If the specified device is associated with an auto dialer, a telephone number must be provided. Use of this option with *systemname* rather than *phone-number* will not give the desired result (see *systemname* below).

**–s** *speed* Specify the transmission speed (300, 1200, 2400, 4800, 9600); The default value is "Any" speed which will depend on the order of the lines in the **Devices** file. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

## USAGE

After making the connection, **cu** runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic XON/XOFF protocol is used to control input from the remote so the buffer is not overrun.

**Commands**

A tilde (~) appearing as the first character of a line is an escape signal which directs **cu** to perform some special action. The *transmit* recognizes the following escape sequences:

| | |
|---|---|
| ~. | Drop the connection and exit (you may still be logged in on the remote machine). |
| ~! | Escape to an interactive shell on the local machine (exiting the shell returns you to **cu**). |
| ~!*cmd* | Run the command *cmd* on the local system (using **sh** –**c**). |
| ~$*cmd* | Run the command *cmd* locally and send its output to the remote system. |
| ~%**cd** | Change the directory on the local system. Note: ~!**cd** will cause the command to be run by a sub-shell, probably not what was intended. |
| ~%**take** *from* [*to*] | Copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places. |
| ~%**put** *from* [*to*] | Copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.<br><br>For both ~%**take** and ~%**put** commands, as each block of the file is transferred, consecutive single digits are printed to the terminal. |
| ~~*line* | Send the line ~*line* to the remote system. |
| ~%**break** | Send a BREAK to the remote system (which can also be specified as ~%**b**). |
| ~%**debug** | Toggle the –**d** debugging option on or off (which can also be specified as ~%**d**). |
| ~**t** | Prints the values of the **termio**(4) structure variables for the user's terminal (useful for debugging). |
| ~**l** | Prints the values of the **termio** structure variables for the remote communication line (useful for debugging). |
| ~%**nostop** | Toggle between XON/XOFF input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the XON or XOFF characters. |

The *receive* process normally copies data from the remote system to its standard output. Internally the program accomplishes this by initiating an output diversion to a file when a line from the remote begins with ~.

Data from the remote is diverted (or appended, if >> is used) to *file* on the local system. The trailing ~> marks the end of the diversion.

The use of ~%**put** requires **stty**(1V) and **cat**(1V) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to these current control characters on the local system. Backslashes are inserted at appropriate places.

The use of ~%**take** requires the existence of **echo**(1V) and **cat**(1V) on the remote system. Also, **tabs** mode (see **stty**(1V)) should be set on the remote system if TAB characters are to be copied without expansion to SPACE characters.

When **cu** is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using ~~. In general, ~ executes the command on the original machine, ~~ executes the command on the next machine in the chain.

**EXAMPLES**

To dial a system whose telephone number is 9 201 555 1212 using 1200 baud (where dialtone is expected after the 9):

```
cu –s1200 9=12015551212
```

If the speed is not specified, "Any" is the default value.

To login to a system connected by a direct line:

**cu –l /dev/ttyXX**

or

**cu –l ttyXX**

To dial a system with the specific line and a specific speed:

**cu –s1200 –l ttyXX**

To dial a system using a specific line associated with an auto dialer:

**cu –l culXX 9=12015551212**

To use a system name:

**cu systemname**

**FILES**

| | |
|---|---|
| **/etc/uucp/Systems** | file listing remote systems |
| **/etc/uucp/Devices** | file listing devices to use |
| **/var/spool/locks/LCK ..*** | lock file to avoid conflicts with UUCP |

**SEE ALSO**

**cat**(1V), **echo**(1V), **stty**(1V), **tip**(1C), **uucp**(1C), **termio**(4)

**DIAGNOSTICS**

Exit code is zero for normal exit, otherwise, 1.

**WARNINGS**

The **cu** command does not do any integrity checking on data it transfers. Data fields with special **cu** characters may not be transmitted properly. Depending on the interconnection hardware, it may be necessary to use a '~.' to terminate the conversion even if **stty 0** has been used. Non-printing characters are not dependably transmitted using either the **~%put** or **~%take** commands.

**BUGS**

There is an artificial slowing of transmission by **cu** during the **~%put** operation so that loss of data is unlikely.

## NAME

cut – remove selected fields from each line of a file

## SYNOPSIS

**cut** –**c** *list* [ *filename* ... ]

**cut** –**f** *list* [ –**d***c* ] [ –**s** ] [ *filename* ... ]

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

Use **cut** to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be of fixed length, (such as on a punched card), or of variable length between lines. They can be marked with a field delimiter character, such as TAB (as specified with the –**d** option). **cut** can be used as a filter; if no files are given, the standard input is used. In addition, a file name of '–' explicitly refers to the standard input.

## OPTIONS

–**c** *list* By character position. *list* is a comma-separated list of integer field numbers (in increasing order), with an optional '–' to indicate ranges:

| | |
|---|---|
| **1,4,7** | characters 1, 4 and 7 |
| **1–3,8** | characters 1 through 3, and 8 |
| **–5,10** | characters (1) through 5, and 10 |
| **3–** | characters 3 through (last) |

–**f** *list* By field position. Instead of character positions, *list* specifies fields that are separated a delimiter (normally a TAB):

**1,4,7** fields 1, 4 and 7

Lines with no field delimiters are normally passed through intact (to allow for subheadings).

–**d***c* Set the field delimiter to *c*. The default is a TAB. Characters with special meaning to the shell such as a TAB or SPACE characters, must be quoted.

–**s** Suppress lines with no delimiter characters.

## EXAMPLES

**cut –d: –f1,5 /etc/passwd**

Mapping of user IDs to names.

**name=who am i | cut –f1 –d" "**

Set **name** to the current login name.

## SEE ALSO

**grep**(1V), **paste**(1V)

## DIAGNOSTICS

**ERROR: line too long**

A line can have no more than 1023 characters or fields.

**ERROR: bad list for c / f option**

Missing –**c** or –**f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

**ERROR: no fields**

The *list* is empty.

**ERROR: no delimiter**

Missing *char* on –**d** option.

**ERROR: cannot handle multiple adjacent backspaces**

Adjacent backspaces cannot be processed correctly.

**WARNING: cannot open** *filename*: *reason*
Either *filename* cannot be read or does not exist. If multiple filenames are present, processing continues.

**WARNING: I/O error reading** *filename*: *reason*
An I/O error occurred when reading *filename*. If multiple filenames are present, processing continues.

## NAME

cxref – generate a C program cross-reference

## SYNOPSIS

**cxref** [ **–c** ] [ **–w** [ *num* ] ] [ **–o** *filename* ] [ **–t** ] *filenames*

## SYSTEM V SYNOPSIS

**/usr/5bin/cxref** [ **–c** ] [ **–w** [ *num* ] ] [ **–o** *filename* ] [ **–t** ] *filenames*

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**cxref** analyzes a collection of C files and attempts to build a cross-reference table. **cxref** utilizes a special option of **cpp**(1) to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the **–c** option, in combination. Each symbol contains an asterisk (*****) before the declaring reference.

## SYSTEM V DESCRIPTION

The System V version of **cxref**, run as **/usr/5bin/cxref**, makes the **C** preprocessor search for include files in **/usr/5include** before searching for them in **/usr/include.**

## OPTIONS

In addition to the **–D**, **–I** and **–U** options (which are identical to their interpretation by **cc**(1V)), the following options are interpreted by **cxref**:

**–c** Print a combined cross-reference of all input files.

**–w**[ *num* ] Width option which formats output no wider than *num* (decimal) columns. This option will default to 80 if *num* is not specified or is less than 51.

**–o** *filename* Direct output to named *file*.

**–s** Operate silently; does not print input file names.

**–t** Format listing for 80-column width.

## FILES

**/usr/tmp/xr***
temporary files

## SEE ALSO

**cc**(1V), **cpp**(1)

## DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files, anyway.

## NOTES

While the compiler allows 8-bit strings and comments, 8-bits are not allowed anywhere else. See **cc**(1V) for an explanation about why **cc** is not 8-bit clean.

## BUGS

**cxref** considers a formal argument in a **#define** macro definition to be a declaration of that symbol. For example, a program that **#includes ctype. h**, will contain many declarations of the variable **c**.

**NAME**

date – display or set the date

**SYNOPSIS**

**date** [ **–u** ] [ **–a** [ – ] *sss.fff* ] [ *yy mm dd hh mm* [ *.ss* ] ] [ *+format* ]

**SYSTEM V SYNOPSIS**

**/usr/5bin/date** [ **–u** ] [ **–a** [ – ] *sss.***fff** ] [ *mm dd hh mm* [ *yy* ] ] [ *+format* ]

**AVAILABILITY**

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

If no argument is given, or if the argument begins with +, **date** displays the current date and time. Otherwise, the current date is set. Only the super-user may **set** the date.

*yy* is the last two digits of the year; the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *.ss* (optional) specifies seconds. The year, month and day may be omitted; the current values are supplied as defaults.

If the argument begins with +, the output of **date** is under the control of the user. The format for the output is similar to that of the first argument to **printf**(3V). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

| | |
|---|---|
| **n** | insert a new-line character |
| **t** | insert a tab character |
| **m** | month of year – 01 to 12 |
| **d** | day of month – 01 to 31 |
| **y** | last 2 digits of year – 00 to 99 |
| **D** | date as mm/dd/yy |
| **H** | hour – 00 to 23 |
| **M** | minute – 00 to 59 |
| **S** | second – 00 to 59 |
| **T** | time as HH:MM:SS |
| **j** | day of year – 001 to 366 |
| **w** | day of week – Sunday = 0 |
| **a** | abbreviated weekday – Sun to Sat |
| **h** | abbreviated month – Jan to Dec |
| **r** | time in AM/PM notation |

**SYSTEM V SYNOPSIS**

When setting the date, the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. The current year is the default if no year is mentioned.

**OPTIONS**

**–u** Display the date in GMT (universal time). The system operates in GMT; **date** normally takes care of the conversion to and from local standard and daylight time. **–u** may also be used to set GMT time.

**–a** [–]*sss.fff* Using the **adjtime**(2) system call, tell the system to slowly adjust the time by *sss.fff* seconds (*fff* represents fractions of a second). This adjustment can be positive or negative. The system's clock will be sped up or slowed down until it has drifted by the number of seconds specified.

**EXAMPLES**

**date 10080045**

Would set the date to Oct 8, 12:45 AM.

If the year were 1986, and the date were so set,

**date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'**

would generate as output:

**DATE: 08/01/86**
**TIME: 14:45:05**

**FILES**

| | |
|---|---|
| **/var/adm/wtmp** | to record time-setting |
| **/usr/share/lib/zoneinfo** | timezone definitions |

**SEE ALSO**

**adjtime**(2), **printf**(3V), **utmp**(5V)

**DIAGNOSTICS**

**date: Failed to set date: Not owner**
If you try to change the date but are not the super-user.

**date: bad format character**
If the field descriptor is not recognizable.

## NAME

dbx – source-level debugger

## SYNOPSIS

**dbx** [ **–f** *fcount* ] [ **–i** ] [ **–I** *dir* ] [ **–k** ] [ **–kbd** ] [ **–P** *fd* ] [ **–r** ] [ **–s** *startup* ] [ **–sr** *tstartup* ] [ *objfile* [ *corefile* | *process-id* ] ]

## AVAILABILITY

This command is available with the *Debugging* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**dbx** is a utility for source-level debugging and execution of programs written in C, or other supported languages such as Pascal and FORTRAN 77. **dbx** accepts the same commands as **dbxtool**(1), but uses a standard terminal (tty) interface.

*objfile* is an object file, produced by **cc**(1V) or another compiler, with the **–g** option to include a symbol table. This symbol table contains the names of all the source files used to create it, and these files are available for perusal while using the debugger.

If no *objfile* is specified, you can use **dbx**'s **debug** command to specify the program to debug.

If there is a file named **core** in the current directory, or a *corefile* argument is specified, you can use **dbx** to examine the state of the program when the core file was produced.

**dbx** commands in the file **.dbxinit** are executed immediately before the symbol table is read, if that file exists in the current directory, or in the user's home directory if **.dbxinit** does not exist in the current directory.

## OPTIONS

**–f** *fcount* Alter the initial estimate of the number functions in the program being debugged. The initial setting is 500.

**–i** Force **dbx** to act as though the standard input were a terminal or terminal emulator.

**–I** *dir* Add *dir* to the list of directories in which to search for a source file. **dbx** normally searches the current directory, and the directory where *objfile* is located. The directory search path can be reset with the **use** command.

**–k** Kernel debugging.

**–kbd** Debug a program that sets the keyboard into up-down translation mode. This flag is necessary if a program uses up-down decoding.

**–P** *fd* Create a pipeline to a **dbxtool**(1) process. *fd* is the file descriptor through which to pipe output to the front-end process. This option is passed automatically to **dbx** by **dbxtool**.

**–r** Execute *objfile* immediately. Parameters follow the object file name (redirection is handled properly). If the program terminates successfully, **dbx** exits. Otherwise, **dbx** reports the reason for termination and waits for a response. **dbx** reads from the terminal (**/dev/tty**) when **–r** is specified and standard input is a file or pipe.

**–s** *startup* Read initialization commands from the file named *startup*.

**–sr** *tstartup* Read initialization commands from the temporary file named *startup*, and then remove that file.

## USAGE

Refer to **dbx** in the *Debugging Tools* manual.

The most useful basic commands to know about are **run**, to run the program being debugged, **where**, to obtain a stack trace with line numbers, **print**, to display variables, and **stop**, to set breakpoints.

**Filenames**

Filenames in **dbx** may include shell metacharacters. The shell used for pattern matching is determined by the SHELL environment variable.

**Expressions**

**dbx expressions** are combinations of variables, constants, procedure calls, and operators. Variables are either variables in the program being debugged or special **dbx** variables whose names begin with **$**. Hexadecimal constants must be preceded by a '0x' and octal constants by a '0'. Character constants must be enclosed in single quotes. Expressions cannot involve strings, structures, or arrays, although elements of structures or arrays may be used.

**Operators**

| | |
|---|---|
| + − * / **div** % | Add, subtract, multiply, divide, integer division, and remainder, respectively. **dbx** has two division operators. '/' always yields a floating-point result and **div** always yields an integral result. |
| << >> & \| ~ | Left-shift, right-shift, bitwise AND, bitwise OR, and bitwise complement. |
| & * | Address of operator, and contents of operator. |
| < > <= >= == != ! | Less than, greater than, less than or equal to, greater than or equal to, equal, not equal, and negation. |
| && \|\| | Logical AND, and logical OR |
| **sizeof** (*cast*) | Size of variable or type and type cast. |
| . −> | Field reference, and pointed-to-field reference (however, dot works for both in **dbx**). |

Precedence and associativity of operators are the same as for C; parentheses can be used for grouping.

If there is no *corefile*, only expressions containing constants are available. Procedure calls require an active child process.

**Scope Rules**

**dbx** uses the current file and function to resolve scope conflicts. Their values are updated as files and functions are entered and exited during execution. You can also change them explicitly by using the **file** and **func** commands. When the current function is changed, the current file is updated along with it, but not vice versa.

**Execution and Tracing Commands**

^C Interrupt. Stop the program being debugged and enter **dbx**.

**run** [ *args* ] [ < *infile* ] [ >| >> *outfile* ]
Start executing *objfile*, reading in any new information from it. With no *args*, use the argument list from the previous **run** command.

| | |
|---|---|
| *args* | Pass *args* as command-line arguments to the program. |
| <\|>\|>> | Redirect input or output, or append output to a file. |

**rerun** [ *args* ] [ < *infile* ] [ >| >> *outfile* ]
Like the **run** command, except that when *args* are omitted, none are passed to the program.

**cont** [ **at** *sourceline* ] [ **sig** *signal* ]
Continue execution from where it stopped.

| | |
|---|---|
| **at** *sourceline* | Start from *sourceline* |
| **sig** *signal* | Continue as if *signal* had occurred. *signal* may be a number or a name as with **catch**. |

**trace** [ **in** *function* ] [ **if** *condition* ]
**trace** *sourceline* [ **if** *condition* ]
**trace** *function* [ **if** *condition* ]
**trace** *expression* **at** *sourceline* [ **if** *condition* ]
**trace** *variable* [ **in** *function* ] [ **if** *condition* ]

Display tracing information. If no argument is specified, each source line is displayed before execution. Tracing is turned off when the function or procedure is exited.

| | |
|---|---|
| **in** *function* | Display tracing information only while executing the function or procedure *function*. |
| **if** *condition* | Display tracing information only if *condition* is true. |
| *sourceline* | Display the line immediately prior to executing it. Source line-numbers from another file are written as *filename:***n.** |
| *function* | Display the routine and source line called from, parameters passed in, and return value. |
| *expression* **at** *sourceline* | Display the value of *expression* whenever *sourceline* is reached. |
| *variable* | Display the name and value whenever *variable* changes. |

**stop at** *sourceline* [ **if** *condition* ]
**stop in** *function* [ **if** *condition* ]
**stop** *variable* [ **if** *condition* ]
**stop if** *condition*

Stop execution when the *sourceline* is reached, *function* is called, *variable* is changed, or *condition* becomes true.

**when in** *function* { *command* ; [ *command* ; ] ... }
**when at** *sourceline* { *command* ; [ *command* ; ] ... }
**when** *condition* { *command* ; [ *command* ; ] ... }

Execute the **dbx** *command*(s) when *function* is called, *sourceline* is reached, or *condition* is true.

**status** [ > *filename* ]

Display active **trace**, **stop** and **when** commands, and associated command numbers.

**delete all**
**delete** *cmd-no* [ , *cmd-no* ] ...

Remove all **traces**, **stops** and **whens**, or those corresponding to each **dbx** *cmd-no* (as displayed by **status**).

**clear** [ *sourceline* ]

Clear all breakpoints at the current stopping point, or at *sourceline*.

**catch** [ *signal* [ , *signal* ] ... ]

Display all signals currently being caught, or catch *signal* before it is sent to the program being debugged. A signal can be specified either by name (with the SIG prefix omitted, as with **kill**(1)) or number. Initially all signals are caught except SIGHUP, SIGEMT, SIGFPE, SIGKILL, SIGALRM, SIGTSTP, SIGCONT, SIGCHLD, and SIGWINCH.

**ignore** [ *signal* [ , *signal* ] ... ]

Display all signals currently being ignored, or stop catching *signal*, which may be specified by name or number as with **catch**.

**step** [*n*]

Execute the next *n* source lines. If omitted, *n* is taken to be 1. Steps into functions.

**next** [*n*]

Execute the next *n* source lines. If omitted, *n* is taken to be 1. **next** steps past functions.

**Naming, Printing and Displaying Data**

Variables from another function or procedure with the same name as one in the current block must be qualified as follows:

[`*sourcefile*`]*function*`*variable*

For Pascal variables there may be more than one *function* or procedure name, each separated by a backquote.

**print** *expression* [ , *expression* ] ...
Print the value of each *expression*, which may involve function calls. Program execution halts when a breakpoint is reached, and **dbx** resumes.

**display** [ *expression* [ , *expression* ] ... ]
Print a list of the expressions currently being displayed, or display the value of each *expression* whenever execution stops.

**undisplay** [ *expression* [ , *expression* ] ... ]
Stop displaying the value of each *expression* whenever execution stops. If *expression* is a constant, it refers to a display-number as shown by the **display** command with no arguments.

**whatis** *identifier*
**whatis** *type*
Print the declaration of the given identifier or type. *type*s are useful to print all the members of a structure, union, or enumerated type.

**which** *identifier*
Print the fully-qualified name of the given identifier.

**whereis** *identifier*
Print the fully qualified name of all symbols matching *identifier*.

**assign** *variable* = *expression*
**set** *variable* = *expression*
Assign the value of *expression* to *variable*. There is no type conversion for operands of differing type.

**set81** *fpreg=word1 word2 word3*
Treat the concatenation of *word1 word2 word3* as a 96-bit, IEEE floating-point value and assign it to the MC68881 floating-point register *fpreg*. (Supported only on Sun-3).

**call** *function*(*parameters*)
Execute the named function. Arguments are passed according to the rules for the source-language of *function*.

**where**[ *n*]
List all, or the top *n*, active functions on the stack.

**dump** [ *function* ]
Display the names and values of local variables and parameters in the current or specified *function*.

**up** [*n*]
**down** [*n*]
Move up (towards "main") or down the call stack, one or *n* levels.

**File Access Commands**

**edit** [ *filename* | *function* ]
Edit the current source file, or the given *filename* or the file that contains *function*.

**file** [ *filename* ]
Print the name of the current source file, or change the current source file to *filename*.

**func** [ *function*| *program* | *objfile* ]
Print the name of the current function, or change to the given *function*, *program*, or *objfile*. Also changes the current scope.

**list** [ *startline* [ , *endline* ] ]
**list** *function*
List the next ten lines from current source file, list from *startline* through *endline*, or and list from five lines above, to five lines below the first line of *function*.

**use** [ *directory-list* ]
Print or set the list of directories in which to search for source files.

**cd** [ *directory* ]
Change the current working directory for **dbx** to *directory* (or to the value of the HOME environment variable).

**pwd** Print the current working directory for **dbx**.

*/reg-exp*[/]
*?reg-exp* [?]
Search the current file for the regular expression *reg-exp*, from the next (previous) line to the end (top). The matching line becomes the new current line.

**Miscellaneous Commands**

**sh** *command-line*
Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

**alias** *new-command-name character-sequence*
Respond to *new-command-name* as though it were *character-sequence*. Special characters occurring in *character-sequence* must be enclosed in quotation marks. Alias substitution as with the C shell (**csh**(1)) also occurs.

**help** [ *command* ]
Display a synopsis of **dbx** commands, or print a short message explaining *command*.

**make** Invoke **make**(1) with the name of the program as its argument. Any arguments set using **dbxenv makeargs** are also passed as arguments.

**setenv** *name string*
Set environment variable *name* to *string*.

**source** *filename*
Read and execute **dbx** commands from *filename*. Useful when the *filename* has been created by redirecting an earlier **status** command.

**quit** Exit **dbx**.

**dbxenv**
**dbxenv case sensitive | insensitive**
**dbxenv fpaasm on | off**
**dbxenv fpabase a[0-7] | off**
**dbxenv makeargs** *string*
**dbxenv stringlen** *num*
**dbxenv speed** *seconds*
Display **dbx** attributes or set the named attribute:

**case** Controls whether upper- and lower-case characters are treated as different values. The default is **sensitive**.

**fpaasm** Controls FPA instruction disassembly. The default is **on**. (Supported only on Sun-3).

**fpabase** Sets the base register for FPA instruction disassembly. The default is **off**. (Supported only on Sun-3 systems).

**makeargs** Sets arguments to pass to **make**. The default is **CC=cc –g**.

**speed** Set the interval between execution during tracing. The default is 0.5 seconds.

**stringlen** Controls the maximum number of characters printed for a "**char** *" variable in a C program. The default is 512.

**debug** [ **–k** ] [ *objfile* [ *corefile* | *pid* ] ]

With no arguments, print the name of the current program. With arguments, stop debugging the current program and begin debugging *objfile* having either *corefile* or the current process ID *pid*. The **–k** option indicates kernel debugging.

**kill** Stop debugging of the current program, but be ready to debug another.

**detach** Detach the current program (process) from **dbx**. **dbx** will be unable to access or modify its state.

**modules**
**modules select** [ **all** | *objfile* ... ]
**modules append** *objfile* [ *objfile* ... ]

The **modules** command displays or sets the current *modules selection list*. If the modules selection list is set, the debugger reads debugging information only for object files in this list. Debugging information for object files not in the modules selection list is ignored.

**modules** with no arguments displays the set of object files for which source level debugging information is currently available, including the path names of any associated source files. If the debugger cannot access a source file for which it has debugging information, it displays the source file name with a trailing '**?**' (question mark) character. Source file path names reflect the current search path as set by the **use** command or the **–I** option.

**modules select** displays the current modules selection list if no *objfile* is given. Otherwise, **modules select** sets the modules selection list to the specified object files. To get complete debugging information, the debugger may need to read object files not in the modules selection list. '**modules select**' displays these "implied" file names with trailing '*' (asterisk) characters (see NOTES). '**modules select all**' discards the modules selection list.

**modules append** appends the specified object files to the modules selection list.

If the modules selection list includes an object file not in the executable being debugged, the debugger issues a warning.

**proc** [ *pid* ]

For kernel debugging. Display which process is mapped into the user area, or map *pid* to the user area.

**Machine-Level Commands**

**tracei** [ *address* ] [ **if** *condition* ]
**tracei** [ *variable* ] [**at** *address* ] [ **if** *condition* ]

Trace execution of a specific machine-instruction address.

**stopi** [ *variable* ] [ **if** *condition* ]
**stopi** [**at** *address* ] [ **if** *condition* ]

Set a breakpoint at a machine instruction address.

**stepi**
**nexti** Single step as in **step** or **next**, but do a single machine instruction rather than a source line.

*address*,*address* / [ *mode* ]
*address* / [*count* ] [ *mode* ]

Display the contents of memory starting at the first (or current) *address* up to the second *address*, or until *count* items have been displayed. The initial display *mode* is **X**. The following modes are supported:

| | |
|---|---|
| **i** | the machine instruction |
| **d** | word in decimal |
| **D** | longword in decimal |
| **o** | word in octal |
| **O** | longword in octal |
| **x** | word in hexadecimal |
| **X** | longword in hexadecimal |
| **b** | byte in octal |
| **c** | byte as a character |
| **s** | strings as characters terminated by a null |
| **f** | single precision real number |
| **F** | double-precision real number |
| **E** | extended-precision real number (not supported on Sun-4) |

An *address* can be specified as an item from the following list, as an expression made up of other addresses and the operators '+', '–', '*', and indirection (unary '*'), or as an arbitrary expression enclosed in parentheses.

| | |
|---|---|
| *&name* | symbolic address |
| *integer* | numeric address |

*address* = [ *mode* ]

Display the value of the *address*.

**Machine Registers**

The machine registers for the current machine type are represented as special **dbx** variables. They can be used in expressions as any other **dbx** variable can. The registers and their variable names are:

*Sun-2 and Sun-3 Registers*

| | |
|---|---|
| **$d[0-7]** | data registers |
| **$a[0-7]** | address registers |
| **$fp** | frame pointer, equivalent to register a6 |
| **$sp** | stack pointer, equivalent to register a7 |
| **$pc** | program counter |
| **$ps** | program status |

*Sun-3-Only Registers*

| | |
|---|---|
| **$fp[0-7]** | MC68881 data registers |
| **$fpc** | MC68881 control register |
| **$fps** | MC68881 status register |
| **$fpi** | MC68881 instruction address register |
| **$fpf** | MC68881 flags register (unused, idle, busy) |
| **$fpa[0-31]** | double-precision interpretation of FPA registers. |
| **$sfpa[0-31]** | single-precision interpretation of FPA registers. |

*Sun-4 Registers*

| | |
|---|---|
| **$g[0-7]** | global registers |
| **$o[0-7]** | "out" registers |
| **$i[0-7]** | "in" registers |
| **$l[0-7]** | "local" registers |
| **$fp** | frame pointer, equivalent to register i6 |
| **$sp** | stack pointer, equivalent to register o6 |

| | |
|---|---|
| **$y** | Y register |
| **$psr** | processor state register |
| **$wim** | window invalid mask register |
| **$tbr** | trap base register |
| **$pc** | program counter |
| **$npc** | next program counter |
| **$f[0-31]** | FPU "f" registers |
| **$fsr** | FPU status register |
| **$fq** | FPU queue |

*Sun386i Registers*

| | |
|---|---|
| **$ss** | stack segment register |
| **$eflags** | flags |
| **$cs** | code segment register |
| **$eip** | instruction pointer |
| **$eax** | general register |
| **$ecx** | general register |
| **$edx** | general register |
| **$ebx** | general register |
| **$esp** | stack pointer |
| **$ebp** | frame pointer |
| **$esi** | source index register |
| **$edi** | destination index register |
| **$ds** | data segment register |
| **$es** | alternate data segment register |
| **$fs** | alternate data segment register |
| **$gs** | alternate data segment register |

Registers for the 80386 lower halves (16 bits) are:

| | |
|---|---|
| **$ax** | general register |
| **$cx** | general register |
| **$dx** | general register |
| **$bx** | general register |
| **$sp** | stack pointer |
| **$bp** | frame pointer |
| **$si** | source index register |
| **$di** | destination index register |
| **$ip** | instruction pointer, lower 16 bits |
| **$flags** | flags, lower 16 bits |

The first four Sun386i 16-bit registers can be split into 8-bit parts:

| | |
|---|---|
| **$al** | lower (right) half of register **$ax** |
| **$ah** | higher (left) half of register **$ax** |
| **$cl** | lower (right) half of register **$cx** |
| **$ch** | higher (left) half of register **$cx** |
| **$dl** | lower (right) half of register **$dx** |
| **$dh** | higher (left) half of register **$dx** |
| **$bl** | lower (right) half of register **$bx** |
| **$bh** | higher (left) half of register **$bx** |

Registers for the 80387 are:

| | |
|---|---|
| **$fctrl** | control register |
| **$fstat** | status register |
| **$ftag** | tag register |
| **$fip** | instruction pointer offset |
| **$fcs** | code segment selector |

| | |
|---|---|
| **$fopoff** | operand pointer offset |
| **$fopsel** | operand pointer selector |
| **$st0 - $st7** | data registers |

**ENVIRONMENT**

**dbx** checks the environment variable EDITOR for the name of the text editor to use with the **edit** command.

**FILES**

| | |
|---|---|
| **core** | default core file |
| **.dbxinit** | local **dbx** initialization file |
| **~/.dbxinit** | user's **dbx** initialization file |

**SEE ALSO**

**cc**(1V), **csh**(1), **dbxtool**(1), **kill**(1), **lex**(1), **make**(1), **yacc**(1)

*Debugging Tools*

**NOTES**

Because the **cc** command does not generate or support 8-bit symbol names, it is inappropriate to make **dbx** 8-bit clean. See **cc**(1V) for an explanation about why **cc** is not 8-bit clean.

To save space, the linker eliminates debugging information redundantly defined in multiple include files. If the linker excluded some of the symbols for an object file in the modules selection list, the debugger must read debugging information from the object files where these symbols were first defined. If these "implied" modules are not in the modules selection list, '**modules select**' displays their names with trailing '*' (asterisk) characters.

**BUGS**

**dbx** does not correctly handle C variables that are local to compound statements. When printing these variables it often gives incorrect results.

**dbx** does not handle FORTRAN entry points well — it treats them as if they were independent routines.

**dbx** does not handle *assigning* to FORTRAN complex types correctly (see the **assign/set** command).

Unlike C, **dbx** does not recognize an array or function name as the address of the array or function. In **dbx**, an array name signifies the entire array, and a function name signifies a call to the function with no arguments. To get the address of an array, take the address of its first element. To get the address of a function, take the address of its name.

Casts do not work with FORTRAN 77 or Pascal.

Executable code incorporated into a source file using an **#include** preprocessor directive confuses **dbx**.

**dbx** is confused by the output of program generators such as **yacc**(1) and **lex**(1).

A step command issued at a procedure call will not work properly when debugging information is available for the function being called and that function is in a shared library.

## NAME

dbxtool – SunView interface for the dbx source-level debugger

## SYNOPSIS

**dbxtool** [ **–d** ] [ **–i** ] [ **–k** ] [ **–kbd** ] [ **–I** *directory* ] [ *objectfile* [ *corefile* ] ]

## AVAILABILITY

This command is available with the *Debugging* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**dbxtool**, a source-level debugger for C, Pascal and FORTRAN 77 programs, is a standard tool that runs within the *SunView* environment. It accepts the same commands as **dbx**, but provides a more convenient user interface.

You can use the mouse to set breakpoints, examine the values of variables, control execution, peruse source files, and so on. **dbxtool** has separate subwindows for viewing source code, entering commands and other uses.

*objectfile* is an object file produced by **cc**(1V), any other Sun compiler, (or a combination of them) with the appropriate flag (**–g**) specified to produce symbol information in the object file. **IMPORTANT:** every stage of the compilation process, including the linking phase, must include the **–g** option. If no *objectfile* is specified, you can use the **debug** command to specify the program to be debugged. The object file contains a symbol table which includes the names of all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

If a file named **core** exists in the current directory or a *corefile* is specified, **dbxtool** can be used to examine the state of the program when it faulted.

Debugger commands in the file **.dbxinit** are executed immediately after the symbolic information is read, if that file exists in the current directory, or in the user's home directory if **.dbxinit** does not exist in the current directory.

## OPTIONS

**–d** Produce debugging information for the pipeline from which it reads **dbx**(1) output.

**–i** Force **dbxtool** to act as though standard input were a terminal.

**–k** Kernel debugging.

**–kbd** Debugs a program that sets the keyboard into up/down translation mode. This flag is necessary if you are debugging a program that uses up/down encoding.

**–I** *directory*
Add *directory* to the list of directories that are searched when looking for a source file. Normally **dbxtool** looks for source files in the current directory and then in the directory where *objectfile* is located. The directory search path can also be set with the **use** command. Multiple **–I** options may be given.

## USAGE

Refer to **dbx**(1) for a summary of **dbx** commands, or *Debugging Tools* for more complete information on using **dbxtool**.

## FILES

| | |
|---|---|
| **core** | default core file |
| **.dbxinit** | local **dbx** initialization file |
| **~/.dbxinit** | user's **dbx** initialization file |

## SEE ALSO

**cc**(1V), **dbx**(1)
*Debugging Tools*
*SunView Programmer's Guide*

**BUGS**

The bugs for **dbx**(1) apply to **dbxtool** as well.
The interaction between scrolling in the *source* subwindow and **dbx**'s regular expression search commands is wrong. Scrolling should affect where the next search begins, but it does not.

## NAME

dc – desk calculator

## SYNOPSIS

**dc** [ *filename* ]

## DESCRIPTION

**dc** is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but an input base, output base, and a number of fractional digits to be maintained may be specified. The overall structure of **dc** is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, and then from the standard input.

Note: **bc**(1) is a preprocessor for **dc** that provides infix (normal arithmetic) notation, a C-like syntax for functions, and reasonable control structures for programs.

The following input constructs are recognized:

### Commands

*number* Push a number onto the stack. A number is an unbroken string of the digits 0-9. It may be preceded by an underscore '_' to input a negative number, and may contain decimal points.

**+ − / * % ^** The top two values on the stack are: added (+), subtracted (−), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack and the result is pushed in their place. Any fractional part of an exponent is ignored.

**s***x* Pop the top of the stack and store into a register named *x*, where *x* is any character.

**S***x* Treat *x* as a stack and push the value onto it.

**l***x* Push the value in register *x* onto the stack. The register *x* is not altered. All registers start with zero value.

**L***x* Treat register *x* as a stack, and pop its top value onto the main stack.

**d** Duplicate the top value on the stack.

**p** Print the top value on the stack. The top value remains unchanged.

**P** Interpret the top of the stack as an ASCII string, remove and print it.

**f** Print all values on the stack and in registers.

**q** Exit the program. If executing a string, pop the recursion level by two.

**Q** Pop the top value on the stack, and pop the string execution level by that value.

**x** Treat the top element of the stack as a character string and execute it as a string of **dc** commands.

**X** Replace the number on the top of the stack with its scale factor.

**[...]** Put the bracketed ASCII string onto the top of the stack.

**<***x* **>***x* **=***x* Pop and compare top two elements of the stack. Execute register *x* if they obey the stated relation.

**v** Replace the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

**!** Interpret the rest of the line as a command.

**c** Clear all values on the stack.

**i** Pop the top value on the stack and use that value as the input radix.

**I** Push the input base on the top of the stack.

**o** Pop the top value on the stack and use as the output radix.

**O** Push the output base on the top of the stack.

**k** The top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

**z** Push the stack level onto the stack.

**Z** Replace the number on the top of the stack with its length.

**?** Take a line of input from the input source (usually the terminal) and execute it.

**; :** Used by **bc** for array operations.

## EXAMPLE

Print the first ten values of n!

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

## SEE ALSO

**bc**(1)

## DIAGNOSTICS

| | |
|---|---|
| **x is unimplemented** | Where x is an octal number. |
| **stack empty** | For not enough elements on the stack to do what was asked. |
| **Out of space** | When the free list is exhausted (too many digits). |
| **Out of headers** | For too many numbers being kept around. |
| **Out of pushdown** | For too many items on the stack. |
| **Nesting Depth** | For too many levels of nested execution. |

## BUGS

Base conversions on fractions are truncated to the number of fractional digits of the input value. The values are not rounded.

## NAME

dd – convert and copy files with various data formats

## SYNOPSIS

**dd** [ *option*=*value* ] ...

## DESCRIPTION

**dd** copies a specified input file to a specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

## OPTIONS

**if**=*name* Input file is taken from *name*; standard input is default.

**of**=*name* Output file is taken from *name*; standard output is default. Note: **dd** creates an explicit output file; therefore the **seek** option is usually useless with explicit output except in special cases such as using magnetic tape or raw disk files.

**ibs**=*n* Input block size *n* bytes (default 512).

**obs**=*n* Output block size *n* bytes (default 512).

**bs**=*n* Set both input and output block size, superseding **ibs** and **obs**; also, if no conversion is specified, it is particularly efficient since no copy need be done. Block sizes for the Sun386i are 9k for 3.5-inch floppy disks, and 126b (blocks) for quarter-inch tapes.

**cbs**=*n* Conversion buffer size.

**skip**=*n* Skip *n* input records before starting copy

**files**=*n* Copy *n* input files before terminating (makes sense only when input is a magtape or similar device).

**seek**=*n* Seek *n* records from beginning of output file before copying. This option generally only works with magnetic tapes and raw disk files and is otherwise usually useless if the explicit output file was named with the **of** option.

**count**=*n* Copy only *n* input records.

**conv**=**ascii** Convert EBCDIC to ASCII.

**ebcdic** Convert ASCII to EBCDIC.

**ibm** Slightly different map of ASCII to EBCDIC.

**block** Convert variable length records to fixed length.

**unblock** Convert fixed length records to variable length.

**lcase** Map alphabetics to lower case.

**ucase** Map alphabetics to upper case.

**swab** Swap every pair of bytes.

**noerror** Do not stop processing on an error.

**sync** Pad every input record to **ibs**.

*arg*, *arg* [,...] Several comma-separated conversions, for a combination of effects. For instance, **conv=sync,block** is useful for reading variable-length output from a pipe.

Where sizes are specified, a number of bytes is expected. A number may end with **k** (kilobytes) to specify multiplication by 1024, **b** (blocks of 512 bytes) to specify multiplication by 512, or **w** (words) to specify multiplication by 4; a pair of numbers may be separated by **x** to indicate a product.

**cbs** is used only if **ascii, unblock, ebcdic, ibm,** or **block** conversion is specified. In the first two cases, **cbs** characters are placed into the conversion buffer, any specified character mapping is done, trailing blanks trimmed and NEWLINE added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer, and blanks added to make up an output record of size **cbs.**

After completion, **dd** reports the number of whole and partial input and output blocks.

**EXAMPLES**

To read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file *x:*

**example% dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase**

Note: the use of raw magtape: **dd** is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

**Sun386i EXAMPLES**

The following write the file *filename* to a 3.5-inch floppy and read from the floppy into a file *filename*, respectively:

**example% dd if=***filename* **of=/dev/rfd0c bs=9k**
**example% dd if=/dev/rfd0c of=***filename* **bs=9k**

Sun386i files names are shown in **fdformat**(1).

**SEE ALSO**

**cp**(1), **fdformat**(1), **tr**(1V)

**DIAGNOSTICS**

*f+p* **records in(out):**
Numbers of full and partial records read(written).

**BUGS**

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The **ibm** conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

The **block** and **unblock** options cannot be combined with the **ascii, ebcdic** or **ibm.** Invalid combinations silently ignore all but the last mutually-exclusive keyword.

## NAME

defaultsedit, defaults_from_input, defaults_to_indentpro, defaults_to_mailrc, indentpro_to_defaults, input_from_defaults, lockscreen_default, mailrc_to_defaults, scrolldefaults, stty_from_defaults – create or edit default settings for SunView utilities

## SYNOPSIS

**defaultsedit** [ *generic-tool-arguments* ]

**defaults_from_input**

**defaults_to_indentpro**

**defaults_to_mailrc**

**indentpro_to_defaults**

**input_from_defaults**

**lockscreen_default**

**mailrc_to_defaults**

**scrolldefaults**

**stty_from_defaults**

## AVAILABILITY

These commands are available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**defaultsedit** is a SunView application that provides a convenient means for inspecting and setting default parameters. It can be viewed as an alternative to the traditional UNIX operating system *.*rc* files that contain initialization options for various commands. Currently, you can use **defaultsedit** to manipulate options to the programs **indent**(1), **mail**(1) and **mailtool**(1), **stty**(1V), and **defaultsedit**, as well as the *menu*, *scrollbar*, *text subwindow* and *tty subwindow* packages, and the SunView environment itself.

The remaining commands are used by **defaultsedit** to perform conversions and other functions; they can also be invoked directly from the shell:

| | |
|---|---|
| **defaults_from_input** | update window-system I/O defaults from current system values |
| **defaults_to_indentpro** | update **indent**(1) defaults in the database from the **.indent.pro** file |
| **defaults_to_mailrc** | update the **.mailrc** file from the defaults database |
| **indentpro_to_defaults** | update **indent** defaults from the **.indent.pro** file |
| **input_from_defaults** | update current system values for window-system I/O from defaults database |
| **lockscreen_default** | apply current default for **lockscreen**(1) display program |
| **mailrc_to_defaults** | update **mail**(1) and/or **mailtool**(1) defaults from the **.mailrc** file |
| **scrolldefaults** | a SunView application that lets you try out different settings from the **Scrollbar** category interactively |
| **stty_from_defaults** | set terminal (TTY) options from defaults database |

Any program or package that a user can customize by setting or changing a parameter could be written so as to get its initialization options from the defaults database. For further information, see *SunView System Programmer's Guide*.

**OPTIONS**

**defaultsedit** accepts all of the generic tool arguments discussed in **sunview**(1).

**USAGE**

This only applies to **defaultsedit.**

**Subwindows**

**defaultsedit** consists of four subwindows. From top to bottom they are:

control — Contains the name of the category currently displayed, and buttons labeled SAVE, QUIT, RESET, and EDIT ITEM. To change the category, click the LEFT mouse button on the word CATEGORY, or use the menu that pops up when you click the RIGHT mouse button.

message — A small text subwindow where messages from **defaultsedit** are displayed.

parameters — Shows all current default parameter names with corresponding values. Clicking the LEFT mouse button over a parameter displays a help string in the message subwindow.

edit — A small text subwindow which enables text editing of parameter values. This is useful for very long text values, such as a long mailing list.

**Control Panel**

SAVE — Save the current values that differ from the standard defaults in your private database — that is, the **.defaults** file in your home directory.

QUIT — Exit without saving any changes.

RESET — Reset the default parameters of the current category to the values in your private database. This is useful if you change some values, then change your mind and want to restore the original values.

EDIT ITEM — Clicking the RIGHT mouse button over the EDIT ITEM button brings up a menu with three choices: COPY ITEM, DELETE ITEM and EDIT LABEL. Only text or numeric items can be edited. Note: edits made using this menu will appear only in your private defaults database, not in the master database. The three editing operations are described below.

COPY ITEM — Choosing COPY ITEM will duplicate the current item. You can then edit both the label and the value of the newly created item. Only items with text or numeric values can be copied in this way. COPY ITEM is useful when you want to change the number of instances of a certain type of item — for example, to insert a new mail alias into your defaults database.

DELETE ITEM — Choosing DELETE ITEM will delete the current item from your private database. It cannot be permanently deleted if the corresponding node is present in the master database. However, you can make it behave like an undefined node by giving it the special value *255Undefined255*.

EDIT LABEL — Choosing EDIT LABEL allows you to edit the label of the current item. When you choose EDIT LABEL, the label of the current item changes from bold to normal face. Then you can select the label and edit it as a normal panel text item.

Note: SunView starts up faster when you set the **Private_only** parameter in the **Defaults** category to **TRUE**, in which case only your private **.defaults** file is read.

**ENVIRONMENT**

DEFAULTS_FILE — The value of this environment variable indicates the file from which private SunView defaults are read. When it is undefined, defaults are read from the **.defaults** file in your home directory.

**FILES**

**/usr/lib/defaults/*.d** System-wide parameters and their standard settings. Each file is a category in **defaultsedit**.

**~/.defaults**

**SEE ALSO**

**indent**(1), **lockscreen**(1), **mail**(1), **mailtool**(1), **stty**(1V), **sunview**(1)

*SunView User's Guide*

*SunView System Programmer's Guide*

**BUGS**

Editing of choice items or categories is not supported by **defaultsedit**. Neither is editing of the master defaults database — to add a new program to the master defaults database, you have to edit a master defaults textfile.

**defaultsedit** reorders mail aliases that appear in the **.mailrc** file. This can adversely affect recursive mail aliases. To avoid this, use the **source** command for **mail**(1) to include a file containing such aliases.

**NAME**

deroff – remove nroff, troff, tbl and eqn constructs

**SYNOPSIS**

**deroff** [ **–w** ] *filename* ...

**AVAILABILITY**

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**deroff** reads each file in sequence and removes all **nroff** and **troff** command lines, backslash constructions, macro definitions, **eqn** constructs (between **.EQ** and **.EN** lines or between delimiters), and table descriptions and writes the remainder on the standard output. **deroff** follows chains of included files (**.so** and **.nx** commands); if a file has already been included, a **.so** is ignored and a **.nx** terminates execution. If no input file is given, **deroff** reads from the standard input file.

**OPTIONS**

**–w** Generate a word list, one word per line. A 'word' is a string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed. All other characters are ignored.

**SEE ALSO**

**eqn**(1), **nroff**(1), **tbl**(1), **troff**(1)

**BUGS**

**deroff** is not a complete **troff** interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output.

**deroff** does not work well with files that use **.so** to source in the standard macro package files.

**NAME**

des – encrypt or decrypt data using Data Encryption Standard

**SYNOPSIS**

**des** **–e** | **–d** [ **–bfs** ] [ **–k** *key* ] [ *input-file* [ *output-file* ] ]

**DESCRIPTION**

**des** encrypts and decrypts data using the NBS Data Encryption Standard algorithm. One of **–e** (for encrypt) or **–d** (for decrypt) must be specified.

The **des** command is provided to promote secure exchange of data in a standard fashion.

Two standard encryption modes are supported by the **des** program, Cipher Block Chaining (CBC — the default) and Electronic Code Book (ECB — specified with **–b**). CBC mode treats an entire file as a unit of encryption, that is, if insertions or deletions are made to the encrypted file then decryption will not succeed. CBC mode also ensures that regularities in clear data do not appear in the encrypted data. ECB mode treats each 8 bytes as units of encryptions, so if parts of the encrypted file are modified then other parts may still be decrypted. Identical values of clear text encrypt to identical values of cipher text.

The key used for the DES algorithm is obtained by prompting the user unless the '**–k** *key*' option is given. If the key is an argument to the **des** command, it is potentially visible to users executing **ps**(1) or a derivative. To minimize this possibility, **des** takes care to destroy the key argument immediately upon entry.

The **des** command attempts to use DES hardware for its job, but will use a software implementation of the DES algorithm if the hardware is unavailable. Normally, a warning message is printed if the DES hardware is unavailable since the software is only about 1/50th as fast. However, the **–f** option will suppress the warning. The **–s** option may be used to force use of software instead of hardware DES.

The **des** command reads from standard input unless *input-file* is specified and writes to standard output unless *output-file* is given.

The following sections give information required to implement compatible facilities in other environments.

Since the CBC and ECB modes of DES require units of 8 bytes to be encrypted, files being encrypted by the **des** command have 1 to 8 bytes appended to them to cause them to be a multiple of 8 bytes. The last byte, when decrypted, gives the number of bytes (0 to 7) which are to be saved of the last 8 bytes. The other bytes of those appended to the input are randomized before encryption. If, when decrypting, the last byte is not in the range of 0 to 7 then either the encrypted file has been corrupted or an incorrect key was provided for decryption and an error message is printed.

The DES algorithm requires an 8 byte key whose low order bits are assumed to be odd-parity bits. The ASCII key supplied by the user is zero padded to 8 bytes and the high order bits are set to be odd-parity bits. The DES algorithm then ignores the low bit of each ASCII character, but that bit's information has been preserved in the high bit due to the parity.

The CBC mode of operation always uses an initial value of all zeros for the initialization vector, so the first 8 bytes of a file are encrypted the same whether in CBC or ECB mode.

**OPTIONS**

**–b** Select ECB (eight bytes at a time) encryption mode.

**–d** Decrypt data.

**–e** Encrypt data.

**–f** Suppress warning message when software implementation is used.

**–s** Select software implementation for the encryption algorithm.

**–k** *key* Use the encryption *key* specified.

**FILES**

**/dev/des?**

**SEE ALSO**

**ps**(1)

**BUGS**

It would be better to use a real 56-bit key rather than an ASCII-based 56-bit pattern. Knowing that the key was derived from ASCII radically reduces the time necessary for a brute-force cryptographic attack.

**RESTRICTIONS**

Software encryption is disabled for programs shipped outside of the U.S. The program will still be able to encrypt files if one can obtain an encryption chip, legally or otherwise.

**NAME**

df – report free disk space on file systems

**SYNOPSIS**

**df** [ **–a** ] [ **–i** ] [ **–t** *type* ] [ *filesystem...* ] [ *filename...* ]

**SYSTEM V SYNOPSIS**

**/usr/5bin/df** [ **–t** ] [ *filesystem...* ] [ *filename...* ]

**AVAILABILITY**

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**df** displays the amount of disk space occupied by currently mounted file systems, the amount of used and available space, and how much of the file system's total capacity has been used. Used without arguments, **df** reports on all mounted file systems, producing something like:

```
example% df
Filesystem  kbytes  used  avail  capacity  Mounted on
/dev/ip0a    7445   4714  1986   70%       /
/dev/ip0g   42277  35291  2758   93%       /usr
```

Note: **used+avail** is less than the amount of space in the file system (kbytes); this is because the system reserves a fraction of the space in the file system to allow its file system allocation routines to work well. The amount reserved is typically about 10%; this may be adjusted using **tunefs**(8). When all the space on a file system except for this reserve is in use, only the super-user can allocate new files and data blocks to existing files. When a file system is overallocated in this way, **df** may report that the file system is more than 100% utilized.

If arguments to **df** are disk partitions (for example, **/dev/ip0a**s or path names, **df** produces a report on the file system containing the named file. Thus '**df .**' shows the amount of space on the file system containing the current directory.

**SYSTEM V DESCRIPTION**

The *System V* version of **df** works in the same manner as above but prints only the amount of available space (in 512 byte units) and the number of free inodes.

**OPTIONS**

**–a** Report on all filesystems including the uninteresting ones which have zero total blocks. (that is, *automounter*)

**–i** Report the number of used and free inodes. Print '*' if no information is available.

**–t** *type* Report on filesystems of a given *type* (for example, **nfs** or **4.2**).

**SYSTEM V OPTIONS**

**–t** Report the total allocated space figures.

**FILES**

**/etc/mtab** List of filesystems currently mounted.

**SEE ALSO**

**du**(1V), **mtab**(5), **quot**(8), **tunefs**(8)

## NAME

diff – display line-by-line differences between pairs of text files

## SYNOPSIS

**diff** [ **–bitw** ] [ **–c** [ # ] | **–e** | **–f** | **–n** | **–h** ] *filename1 filename2*
**diff** [ **–bitw** ] [ **–D***string* ] *filename1 filename2*
**diff** [ **–bitw** ] [ **–c** [ # ] | **–e** | **–f** | **–n** | **–h** ] [ **–l** ] [ **–r** ] [ **–s** ] [ **–S***name* ] *directory1 directory2*

## DESCRIPTION

**diff** is a *differential* file comparator. When run on regular files, and when comparing text files that differ during directory comparison (see the notes below on comparing directories), **diff** tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, **diff** finds a smallest sufficient set of differences. If neither *filename1* nor *filename2* is a directory, either may be given as '–', in which case the standard input is used. If *filename1* is a directory, a file in that directory whose filename is the same as the filename of *filename2* is used (and vice versa).

There are several options for output format; the default output format contains lines of these forms:

*n1* **a** *n3*,*n4*
*n1*,*n2* **d** *n3*
*n1*,*n2* **c** *n3*,*n4*

These lines resemble **ed**(1) commands to convert *filename1* into *filename2*. The numbers after the letters pertain to *filename2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *filename2* into *filename1*. As in **ed**(1), identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by '<', then all the lines that are affected in the second file flagged by '>'.

If both arguments are directories, **diff** sorts the contents of the directories by name, and then runs the regular file **diff** program as described above on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed.

## OPTIONS

**–b** Ignore trailing blanks (SPACE and TAB characters) and treat all other strings of blanks as equivalent.

**–i** Ignore the case of letters; for example, '**A**' will compare equal to '**a**'.

**–t** Expand TAB characters in output lines. Normal or **–c** output adds character(s) to the front of each line which may alter the indentation of the original source lines and make the output listing difficult to interpret. This option will preserve the original source's indentation.

**–w** Ignore all blanks (SPACE and TAB characters); for example, '**if ( a = = b )**' will compare equal to '**if(a= =b)**'.

The following four options are mutually exclusive:

**–c**[#] Produce a listing of differences with lines of context. The default is to present 3 lines of context and may be changed, (to 10, for example), by **–c10**. With **–c** the output format is modified slightly: output begins with identification of the files involved and their creation dates, then each change is separated by a line with a dozen * s. The lines removed from *filename1* are marked with '– '; those added to *filename2* are marked '+ '. Lines which are changed from one file to the other are marked in both files with '! '.

Changes which lie within <context> lines of each other are grouped together on output. This is a change from the previous '**diff –c**' but the resulting output is usually much easier to interpret.

**–e** Produce a script of **a**, **c**, and **d** commands for the editor **ed**, which will recreate *filename2* from *filename1*.

In connection with **–e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version **ed** scripts ($2, $3, ...) made by **diff** need be on hand. A 'latest version' appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed – $1
```

Extra commands are added to the output when comparing directories with **–e**, so that the result is a **sh** script for converting text files which are common to the two directories from their state in *directory1* to their state in *directory2*.

**–f** Produce a script similar to that of **–e**, not useful with **ed**, which is in the opposite order.

**–n** Produce a script similar to that of **–e**, but in the opposite order and with a count of changed lines on each insert or delete command.

**–h** Do a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

Options for the second form of **diff** are as follows:

**–D***string*
Create a merged version of *filename1* and *filename2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *filename1*, while defining *string* will yield *filename2*.

Options when comparing directories are:

**–l** Long output format; each text file **diff** is piped through **pr**(1V) to paginate it, other differences are remembered and summarized after all text file differences are reported.

**–r** Apply **diff** recursively to common subdirectories encountered.

**–s** Report files which are the same, which are otherwise not mentioned.

**–S***name*
Start a directory **diff** in the middle, beginning with file *name*.

## ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **diff**. On entry to **diff**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## FILES

**/tmp/d?????**
**/usr/lib/diffh** for **–h**

## SEE ALSO

**cc**(1V), **cmp**(1), **comm**(1), **cpp**(1), **diff3**(1V), **ed**(1), **pr**(1V), **locale**(5), **iso_8859_1**(7)

## DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

**Missing newline at end of file***X*
Indicates that the last line of file *X* did not have a NEWLINE. If the lines are different, they will be flagged and output, although the output will seem to indicate they are the same.

## BUGS

Editing scripts produced under the **–e** or **–f** option are naive about creating lines consisting of a single '.'.

When comparing directories with the **–b**, **–w**, or **–i** options specified, **diff** first compares the files (as in **cmp**(1), and then runs the regular **diff** algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string or case differences.

The **–D** option ignores existing preprocessor controls in the source files, and can generate **#ifdefs**'s with overlapping scope. The output should be checked by hand, or run through '**cc –E**' (see **cc**(1V)) and then **diff**ed with the original source files. Discrepancies revealed should be corrected before compilation.

**NAME**

diff3 – display line-by-line differences between 3 files

**SYNOPSIS**

**diff3** [ **–exEX3** ] *filename1 filename2 filename3*

**SYSTEM V SYNOPSIS**

**/usr/5bin/diff3** [ **–ex3** ] *filename1 filename2 filename3*

**AVAILABILITY**

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**diff3** compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

| | |
|---|---|
| ==== | All three files differ |
| ====**1** | *filename1* is different |
| ====**2** | *filename2* is different |
| ====**3** | *filename3* is different |

The types of differences between a given range within the given files are indicated in one of these ways:

| | |
|---|---|
| *f*: *n1* **a** | Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3. |
| *f*: *n1* , *n2* **c** | Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*. |

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

**OPTIONS**

The options to **diff3** instruct it to produce a script for the editor **ed**, rather than a list of differences. This script will incorporate some or all of the differences between *filename2* and *filename3* into *filename1*. This script will not include a **w** or **q** command at the end, so that it will not write out the changed file.

| | |
|---|---|
| **–e** | Produce a script that will incorporate all changes between *filename2* and *filename3*, that is, the changes that normally would be flagged '====' and '====**3**'. |
| **–x** | Produce a script that will incorporate only changes flagged '====**'**. |
| **–3** | Produce a script that will incorporate only changes flagged '====**3**'. |
| **–E** | Produce a script that will incorporate all changes between *filename2* and *filename3*, but treat overlapping changes (that is, changes that would be flagged with ==== in the normal listing) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by **<<<<<<** and **>>>>>>** lines. |
| **–X** | Produce a script that will incorporate only changes flagged ====, but treat these changes in the manner of the **–E** option. |

For example, suppose lines 7-8 are changed in both *filename1* and *filename2*. Applying the edit script generated by the command

**diff3 –E** *filename1 filename2 filename3*

to *filename1* results in the following file.

**lines 1-6**
**of** *filename1*
**<<<<<<<** *filename1*
**lines 7-8**
**of filename1**
=======
**lines 7-8**
**of** *filename3*
**>>>>>>>** *filename3*
**rest of** *filename1*

**SYSTEM V OPTIONS**

The System V version of **diff3** does not support the **–E** and **–X** options. The script produced by the **–e**, **–x**, and **–3** options *does* include a **w** and **q** command at the end, so that it *will* write out the changed file.

**EXAMPLES**

The following command will incorporate all the changes between *filename2* and *filename3* into *filename1*, and print the resulting file to the standard output. If the System V version of **diff3**, is used, *filename1* will be replaced with the resulting file.

**(diff3 –e** *filename1 filename2 filename3*; **echo '1,$p') | ed –** *filename1*

**FILES**

**/tmp/d3?????**
**/usr/lib/diff3**
**/usr/5lib/diff3prog**

**SEE ALSO**

**diff**(1), **ed**(1)

**BUGS**

Text lines that consist of a single '.' will defeat a **–e** option.

**NAME**

diffmk – mark differences between versions of a troff input file

**SYNOPSIS**

**diffmk** *oldfile newfile markedfile*

**AVAILABILITY**

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**diffmk** compares two versions of a file and creates a third version that includes "change mark" (**.mc**) commands for **nroff**(1) and **troff**(1). *oldfile* and *newfile* are the old and new versions of the file. **diffmk** generates *markedfile*, which, contains the text from *newfile* with **troff**(1) "change mark" requests (**.mc**) inserted where *newfile* differs from *oldfile*. When *markedfile* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

**diffmk** can also be used in conjunction with the proper **troff** requests to produce program listings with marked changes. In the following command line:

**diffmk old.c new.c marked.c ; nroff reqs marked.c | pr**

the file **reqs** contains the following **troff** requests:

```
.pl 1
.ll 77
.nf
.eo
.nh
```

which eliminate page breaks, adjust the line length, set no-fill mode, ignore escape characters, and turn off hyphenation, respectively.

If the characters | and * are inappropriate, you might run *markedfile* through **sed**(1V) to globally change them.

**SEE ALSO**

**diff**(1), **nroff**(1), **sed**(1V), **troff**(1)

**BUGS**

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, that is, replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

**NAME**

dircmp – compare directories

**SYNOPSIS**

**/usr/5bin/dircmp** [ **–d** ] [ **–s** ] [ **–w***n* ] *dir1 dir2*

**AVAILABILITY**

This command is available with the System V software installation option. See *Installing SunOS 4.1* for information on how to install this command.

**DESCRIPTION**

**dircmp** examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the filenames common to both directories have the same contents.

**OPTIONS**

**–d** Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in **diff**(1).

**–s** Suppress messages about identical files.

**–w***n* Change the width of the output line to *n* characters. The default width is 72.

**SEE ALSO**

**cmp**(1), **diff**(1)

*Installing SunOS 4.1*

**NAME**

dis – object code disassembler for COFF

**SYNOPSIS**

**dis** [ **–o** ] [ **–V** ] [ **–L** ] [ **–d** *sec* ] [ **–da** *sec* ] [ **–F** *function* ] [ **–t** *sec* ] [ **–l** *string* ] *coff-file* ...

**AVAILABILITY**

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

**DESCRIPTION**

The **dis** command produces an assembly-language listing of *coff-file*, which may be any object file in COFF format, or an archive of COFF object files.

The listing includes assembly statements and an octal or hexadecimal representation of the binary that produced those statements.

**OPTIONS**

**–o** Print numbers in octal. The default is hexadecimal.

**–V** Print, on standard error, the version number of the disassembler being executed.

**–L** Lookup source labels in the symbol table for subsequent printing. This option works only if the file was compiled with additional debugging information (e.g., the **–g** option of **cc**(1V)).

**–d** *sec* Disassemble the named section as data, printing the offset of the data from the beginning of the section.

**–da** *sec* Disassemble the named section as data, printing the actual address of the data.

**–F** *function*
Disassemble only the named function in each object file specified on the command line. The **–F** option may be specified multiple times on the command line.

**–t** *sec* Disassemble the named section as text.

**–l** *string*
Disassemble the library file specified by *string*. For example, **dis –l x –l z** disassembles **libx.a** and **libz.a**. All libraries are assumed to be in **/usr/lib**.

If the **–d**, **–da** or **–t** options are specified, only those named sections from each user-supplied file name will be disassembled. Otherwise, all sections containing text will be disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as **[5]**, represents that the breakpointable line number starts with the following instruction. These line numbers will be printed only if the file was compiled with additional debugging information (e.g., the **–g** option of **cc**(1V)). An expression such as **<40>** in the operand field or in the symbolic disassembly, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A function name will appear in the first column, followed by ( ).

**FILES**

**/usr/lib**

**SEE ALSO**

**cc**(1V) **coff**(5)

**NOTES**

Because the assembler does not generate or support 8-bit symbol names, it is inappropriate to make **dis** 8-bit clean. See **as**(1).

## NAME

domainname – set or display name of the current NIS domain

## SYNOPSIS

**domainname** [ *name-of-domain* ]

## DESCRIPTION

Without an argument, **domainname** displays the name of the current domain, which typically encompasses a group of hosts under the same administration. As such, the name of a Network Information Service (NIS) domain is normally also a valid Internet domain name, and can be used in conjunction with the **sendmail**(8) and the name server **named**(8C).

Only the super-user can set the name of the domain, by giving **domainname** an argument; this can be done only in the startup script **/etc/rc.local**.

## SEE ALSO

**named**(8C), **sendmail**(8), **ypinit**(8)

## NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

## NAME

dos – SunView window for IBM PC/AT applications

## SYNOPSIS

**dos** [ **–b** ] [ **–p** *config* ] [ **–q** ] [ **–s** ] [ **–update** *time* ] [ **–w** ] [ **–c** *command* ]

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

A window created by **dos** looks and acts like the screen of an IBM PC/AT or compatible computer running MS-DOS 3.3, except that it has expanded features. It allows sharing of files with the SunOS operating system, copying and pasting data between windows, and piping and redirection. You may run any reasonable number of DOS windows simultaneously.

Shrinking or expanding the window will not change the contents to accommodate the new size.

## USAGE

### Menu

The menu available in the window by pressing the right mouse button allows various controls over the work in the window. **Edit** allows you to copy and paste between windows. The **Show Screen** menu item selects the type of screen display—either Hercules, CGA, or Monochrome. Use the DOS MODE command to set the corresponding DOS display mode. See the *Sun386i User's Guide* or on-line help for more information. The **Mouse** menu item allows you to control whether the mouse operates like a Microsoft or compatible mouse or in normal SunView fashion (see *Sun386i Advanced Skills* for instructions on enabling Microsoft mouse driver software). The **Send to printer** menu item allows you to send queued jobs to the print spooler. **Sound** controls the volume of sounds from the DOS window. **Device** allows you to select which disks and other devices will be used and which are to be considered write only. The **Reboot DOS Window** item is equivalent to restarting the window. This can also be accomplished by pressing the CONTROL, ALT, and DELETE keys simultaneously.

### Printer Assignments

DOS uses three printer designations: **LPT1**, **LPT2**, and **LPT3**. The default settings are: files sent to **LPT1** go to the default system printer. Files sent to **LPT2** are appended to the file **lpt-2** in your home directory. Epson-compatible print jobs can be sent to **LPT3** to yield Epson FX-80 quality output on a Postscript printer.

### Drives

The DOS command **FORMAT A: /S** works only if the current working directory contains DOS files. This is usually Drive C and sometimes Drive A.

| | |
|---|---|
| Drive A | The Sun386i 3.5-inch diskette drive, used for reading PC format diskettes onto the hard disk and writing data to be stored on floppy. Drive A is not accessible across a network. |
| Drive B | An optional 5.25-inch diskette drive. Same restrictions as Drive A. |
| Drive C | A virtual disk stored in the **~/pc/C:** file. Files written to Drive C cannot be accessed from the SunOS operating system. Drive C is generally intended for storage of applications and copy protected software but not data. To DOS, drive C is a 20-megabyte drive. You can install copy-protected software on drive C, but not on other drives. |
| Drives D through S | Equivalents of the SunOS operating system directories. They can be accessed from either the DOS or SunOS operating systems, and can contain any number of files and other directories. The SunOS directories referenced by DOS drives other than D, H, and R (described below) are user-defined (using the **DOS EXTEND** command). |

Drive D The current SunOS directory when the DOS window was opened. May subsequently be changed to any other directory.

Drive H The home directory of the user who opened the window. May subsequently be changed to any directory in the user's home directory tree.

Drive R Initially equivalent to the root directory of the SunOS operating system.

**File Sharing between SunOS and DOS**

File names under DOS consist of 8 characters, a period, and a 3 character extension. When a SunOS filename does not comply with these rules, its name is modified by placing a tilde (~) in an appropriate location so that the file name conforms to DOS specifications while remaining unique. It is recommended that filenames conform to DOS requirements for files to be used in both the SunOS and DOS operating systems.

Because the SunOS and DOS operating systems use different conventions for RETURN characters, **dos2unix** and **unix2dos** are provided to convert text files between the two formats.

**Command Sharing between SunOS and DOS**

The **/etc/dos/unix** directory contains a list of SunOS commands accessible from DOS. Other SunOS commands not in this list can be executed from DOS with the command '**unix** *command*'. SunOS commands always use SunOS filename conventions and DOS commands always use DOS filename conventions, regardless of whether either type of command is executed from the SunOS or DOS operating system. Only DOS commands can use drives A and C.

## OPTIONS

**–b** Boots (loads) DOS and opens a window using the **AUTOEXEC.BAT** and **CONFIG.SYS** files instead of **~/pc/.quickpc**. A DOS sign-on message is displayed in the window.

**–s** Boot DOS and save a new **.quickpc** unless **C:AUTOEXEC.BAT**, **C:CONFIG.SYS**, or **/etc/dos/defaults/rom** has a date newer than the **.quickpc** file (see the **–s** option).

**–p** *config*
Loads an alternate file instead of **setup.pc**.

**–q** Forces **dos** to read settings from the **.quickpc** file (as specified in **setup.pc**) even if **C:AUTOEXEC.BAT**, **C:CONFIG.SYS**, or **/etc/dos/defaults/rom** have been updated since you last typed **dos –s**.

**–s** Boot DOS and save a new **.quickpc** file under the name specified on the SAVE line in **~/pc/setup.pc**. Use this option after making changes to drive C's **AUTOEXEC.BAT** or **CONFIG.SYS**. Exits DOS after saving the **.quickpc** file.

**–update**
Gives you a new drive C and a new **setup.pc** using the settings from **/etc/dos/defaults/C:** and **/etc/dos/defaults/setup.pc**, respectively.

**–w** Runs DOS text-only commands and applications in the current SunView Commands window.

**–c** *command*
Executes the given DOS command in the newly created window. If you use the **–c** option, **–c** and the command that follows it must be the last items on the command line.

## ENVIRONMENT

DOS_LOCKING This environment variable determines which locking service is used to lock drive C for write access. If it is set to on, DOS uses the locking service on the server where the home directory is located. This locks drive C for access from any DOS window on the network. If it is set to off, DOS uses the local system's locking service. This locks drive C only for access from DOS windows running on the local system. The default is on. Some servers (for example, some VAX/Ultrix systems) do not provide an NFS locking service. For home directories stored on these servers, set the variable to off to avoid an error message when a DOS window starts up.

DOS_PRINTER The value of this environment variable indicates the timeout (in seconds) for printing. A value of 20 (the default) indicates that jobs will be sent to the UNIX print spooler after 20 seconds of no printing activity from DOS to that printer. A value of 0 indicates that the spooler must be flushed manually from the menu in the window.

DOSLOOKUP

If on, this environment variable indicates that a command should be tried as a DOS command if not recognized by the SunOS system. If DOS supports the command, a DOS window is created and the command executed in that window. If the command does not exist, the normal SunOS error message results.

**FILES**

| | |
|---|---|
| **/etc/dos/unix** | Files in this directory indicate which SunOS commands are accessible from DOS. |
| **/etc/dos/defaults/.quickpc** | Default **.quickpc** file copied into a user's home PC directory (**~/pc**) the first time a DOS window is started. Not used by DOS in this location. |
| **/etc/dos/defaults/setup.pc** | Default **setup.pc** file copied into a user's home DOS directory (**~/pc**) the first time a DOS window is started. Not used by DOS in this location. |
| **/etc/dos/defaults/boards.pc** | Stores information about IBM PC/XT/AT-compatible boards installed in your system. |
| **/etc/dos/defaults/C:** | Default drive C file copied into a user's home PC directory the first time a DOS window is started. |
| **~/pc/autoexec.bat** | Contains drive assignments, search paths, and other startup commands. Searched after **C:AUTOEXEC.BAT** and **D:AUTOEXEC.BAT**. |
| **C:AUTOEXEC.BAT** | Contains commands to access system printers and special drives. You should not need to change the **AUTOEXEC.BAT** on drive C. Put your changes in the **AUTOEXEC.BAT** on drive H (in your home directory). **C:AUTOEXEC.BAT** is not accessible from the SunOS system. |
| **D:AUTOEXEC.BAT** | If an **AUTOEXEC.BAT** file exists in the current directory, DOS tries execute faster running **C:AUTOEXEC.BAT**. |
| **C:CONFIG.SYS** | Specifies device drivers and other system parameters. **C:CONFIG.SYS** is not accessible from the SunOS system. |
| **~/pc/setup.pc** | Defines printers, standard PC devices, and drive C. One or more of these files may exist, under various names which you assign. |
| **~/pc/.quickpc** | An image of DOS as last saved with **dos -s**, including all DOS environment variables and drivers that were in effect at that time. DOS normally reads this file at startup. |
| **~/pc/C:** | A user's personal copy of drive C. |

**DIAGNOSTICS**

**Cannot save** *filename* **quick-start file.**

The **dos** command was unable to save the specified quick-start file. Check the SAVE setting in your PC setup file (normally **~/pc/setup.pc**). Also check file access permissions on the specified quick-start file.

**Cannot load** *filename* **quick-start file.**

**dos** was unable to read the specified quick-start file. Check the SAVE setting in your **setup.pc** file. Also check file access permissions on the specified quick-start file.

**Possible software incompatibility. Unsupported 286 instruction** *instruction* **at** *address*.
**Possible software incompatibility. Unsupported 386 instruction.**
**Possible software incompatibility. Segment wrap.**
**Possible software incompatibility. Two-byte opcode.**

The application you are running was written specifically for 80286 or 80386 machines. Software run from a DOS window must be compatible with 8086 systems.

**Copying default configuration files into your home directory.**
This is the first time you have run the **dos** command. A ~/**pc** directory is being set up, and DOS-related files are being copied into it.

**Another DOS window already has access to** *device*

**IRQ level** *number* **is still in use by another DOS window.**
Your PC configuration file (normally ~/**pc/setup.pc**) is requesting access to a physical device that another DOS window is using.

**Port number** *number* **out of range for** *board* **board.**
The port number specified in the **/etc/dos/defaults/boards.pc** is invalid.

**IRQ value** *number* **out of range for** *board* **board.**
The interrupt level specified in the **/etc/dos/defaults/boards.pc** is invalid.

**IRQ level** *number* **is in use by a Unix driver.**
There is a Unix driver servicing the board you are trying to attach to DOS. You are using the wrong IRQ level or you should use the driver instead.

**Interrupt level** *number* **is used by DOS to support** *device*
The interrupt level specified in the **/etc/dos/defaults/boards.pc** conflicts with an interrupt value currently being used by either a physical or emulated DOS device.

**I/O address range** *address-address* **requested for** *name* **board already in use by** *device*.
The address range specified in the **/etc/dos/defaults/boards.pc** conflicts with range currently being used by either a physical or emulated DOS device.

**Cannot share** *device* **with a hardware interrupt or DMA channel.**
A shared device specified in the **/etc/dos/defaults/boards.pc** was also assigned an interrupt level in this file. Shared devices cannot be assigned interrupt levels.

**Couldn't find** *name* **board in boards.pc.**
A file specified in the PC setup file (normally ~/**pc/setup.pc**) is not listed in the **/etc/dos/defaults/boards.pc** file. Check the **setup.pc** file, or add an entry for the board in **boards.pc**.

**ROM is newer than .quickpc. Rebooting** *program_name*.
Save a new **.quickpc** file by issuing the command **dos –s**.

**Warning: Your personal drive C (***pathname***) is not protected against simultaneous access by more than one workstation. Ask your system administrator to upgrade** *server* **to use the lock manager. Until your home directory server is updated with this program, do not use** *program_name* **when you are logged into more than one workstation.**

The system on the network where your drive C is stored has not protected the drive against access by DOS windows in other workstations on the network. This usually means that the server where your home directory is stored does not provide an NFS locking service. To avoid this error message, set the environment variable DOS_LOCKING to off.

## SEE ALSO

**dos2unix(1), unix2dos(1)**
*Sun386i User's Guide*
*Sun386i Advanced Skills*
*Sun MS-DOS Reference Manual*

## NAME

dos2unix – convert text file from DOS format to ISO format

## SYNOPSIS

**dos2unix** [ **–ascii** ] [ **–iso** ] [ **–7** ] *originalfile convertedfile*

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*.x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

**dos2unix** converts characters in the DOS extended character set to the corresponding ISO standard characters.

This command can be invoked from either DOS or SunOS. However, the filenames must conform to the conventions of the environment in which the command is invoked.

If the original file and the converted file are the same, **dos2unix** will rewrite the original file after converting it.

## OPTIONS

**–ascii** Removes extra carriage returns and converts end of file characters in DOS format text files to conform to SunOS requirements.

**–iso** This is the default. It converts characters in the DOS extended character set to the corresponding ISO standard characters.

**–7** Convert 8 bit DOS graphics characters to 7 bit space characters so that SunOS can read the file.

## DIAGNOSTICS

**File *filename* not found, or no read permission**
The input file you specified does not exist, or you do not have read permission (check with the SunOS **ls –l** command).

**Bad output filename *filename*, or no write permission**
The output file you specified is either invalid, or you do not have write permission for that file or the directory that contains it. Check also that the drive or diskette is not write-protected.

**Error while writing to temporary file**
An error occurred while converting your file, possibly because there is not enough space on the current drive. Check the amount of space on the current drive using the DIR command. Also be certain that the default diskette or drive is write-enabled (not write-protected). Note that when this error occurs, the original file remains intact.

**Could not rename temporary file to**
**Translated temporary file name = *filename*.**
The program could not perform the final step in converting your file. Your converted file is stored under the name indicated on the second line of this message.

## SEE ALSO

**dos**(1), **unix2dos**(1)

*Sun386i Advanced Skills*
*Sun MS-DOS Reference Manual*

## NAME

**du** – display the number of disk blocks used per directory or file

## SYNOPSIS

**du** [ **–s** ] [ **–a** ] [ *filename* ... ]

## SYSTEM V SYNOPSIS

**/usr/5bin/du** [ **–s** ] [ **–a** ] [ **–r** ] [ *filename* ... ]

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**du** gives the number of kilobytes contained in all files and, recursively, directories within each specified directory or file *filename*. If *filename* is missing, '.' (the current directory) is used.

A file which has multiple links to it is only counted once.

## SYSTEM V DESCRIPTION

The System V version of **du** gives the number of 512-byte blocks rather than the number of kilobytes.

## OPTIONS

**–s** Only display the grand total for each of the specified *filename*s.

**–a** Generate an entry for each file.

Entries are generated only for each directory in the absence of options.

## SYSTEM V OPTIONS

**–r** The System V version of **du** is normally silent about directories that cannot be read, files that cannot be opened, etc. The **–r** option will cause **du** to generate messages in such instances.

## EXAMPLE

Here is an example of using **du** in a directory. We used the **pwd**(1) command to identify the directory, then used **du** to show the usage of all the subdirectories in that directory. The grand total for the directory is the last entry in the display:

```
% pwd
/usr/ralph/misc
% du
5       ./jokes
33      ./squash
44      ./tech.papers/lpr.document
217     ./tech.papers/new.manager
401     ./tech.papers
144     ./memos
80      ./letters
388     ./window
93      ./messages
15      ./useful.news
1211    .
%
```

## SEE ALSO

**df**(1V), **pwd**(1), **quot**(8)

## BUGS

Filename arguments that are not directory names are ignored, unless you use **–a** .
If there are too many distinct linked files, **du** will count the excess files more than once.

**NAME**

echo – echo arguments to the standard output

**SYNOPSIS**

**echo** [ **–n** ] [ *argument* ... ]

**SYSTEM V SYNOPSIS**

**/usr/5bin/echo** *argument* ...

**AVAILABILITY**

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**echo** writes its arguments on the standard output. Arguments must be separated by SPACE characters or TAB characters, and terminated by a NEWLINE.

**echo** is useful for producing diagnostics in shell programs and for writing constant data on pipes. If you are using the Bourne shell (**sh**(1)), you can send diagnostics to the standard error file by typing:

**echo ... >&2**

**SYSTEM V DESCRIPTION**

Note: If **/usr/5bin** is ahead of **/usr/bin** in the Bourne shell's search path, its built-in **echo** command mimics the System V version of **echo** as described here.

**echo** also understands C –like escape conventions; beware of conflicts with the shell's use of '\':

| | |
|---|---|
| **\b** | BACKSPACE |
| **\c** | Print line without NEWLINE |
| **\f** | FORMFEED |
| **\n** | NEWLINE |
| **\r** | RETURN |
| **\t** | TAB |
| **\v** | vertical TAB |
| **** | backslash |
| *****n* | the 8-bit character whose ASCII code is the 1-, 2-, 3- or 4-digit octal number *n*. The first digit must be zero. |

**OPTIONS**

**–n** Do not add the NEWLINE to the output.

**SEE ALSO**

**sh**(1)

## NAME

ed, red – basic line editor

## SYNOPSIS

**ed** [ – ] [ **–sx** ] [ **–p** *string* ] [ *filename* ]

**red** [ – ] [ **–sx** ] [ **–p** *string* ] [ *filename* ]

## DESCRIPTION

**ed** is the most basic line editor of the UNIX system. Although superseded by **ex**(1) and **vi**(1) for most purposes, **ed** is still used by various system utilities.

**ed** operates on a copy of *filename*, called a buffer, and overwrites a file only when you issue the **w** (write) command. **ed** provides line oriented editing commands to display or change lines, to insert and delete lines from the buffer, to move or copy lines within the buffer, or to substitute character strings within lines.

**red** is a restricted version of **ed**. It will only allow editing of files in the current directory, and prohibits executing commands using **!***command*. Attempts to bypass these restrictions result in an error message.

## OPTIONS

–

**–s** Suppress printing of character counts (by **e**, **r**, and **w** commands), diagnostics (by **e** and **q** commands), and the **!** prompt (after a **!** command). Also, suppress printing the **?** diagnostic before overwriting unsaved changes in the buffer.

**–x** Edit an encrypted file (see **crypt**(1) for details).

**–p** *string* Use *string* as the editing prompt in command mode.

## USAGE

### Command Structure

**ed** commands have a simple and regular structure. They consist of an optional *address*, or two optional *address*es separated by a comma or semicolon, then a single-character *command*, which may be followed by a *parameter* for that *command*:

[*address*[ ,*address* ]] *command* [ *parameter* ]

If only one *address* is specified, operations are performed on that line. If two *address*es are specified, **ed** performs the operation on the inclusive range of lines. Commands that requires an *address* use certain addresses by default, typically the address of the current line.

For example, **1,10p** means "print (display) lines 1 through 10" (two addresses), **5a** means "append text after line 5" (one address), and **d** means "delete the current line" (no address with the current line used as default). The meaning of *parameter* varies for each operation — for the move (**m**) and transfer (**t**) operations, for instance, it is the line that the addressed lines are to be moved to or transferred after. For reading (**r**) and writing (**w**) a file, *parameter* specifies the name of the file that is to be read or written.

**ed** is extremely terse in its interaction with the user. Its normal response to most problems is simply a question mark (**?**). This may happen when **ed** cannot find a specified line in the buffer, or if a search for a regular expression fails in a substitute (**s**) command. The **h** command prints a somewhat more complete diagnostic for the most recent error encountered; the **H** command requests that the diagnostic be printed for all errors.

### Addresses

Lines can be addressed in several ways:

*nnn* By line number. Lines in the buffer are numbered relative to the start of the buffer. When displayed, line numbers are not physically present with the text of the file or buffer.

**$** The last line of the buffer.

**.** The current line. **ed** keeps track of the line on which you last performed an operation. This line is called the *current line*. You can address this line by typing a "dot" character.

±*n* By relative line number. Address the line number that is *n* lines higher, or *n* lines lower than the current line.

'*c* Address the line marked with the mark character *c*, which must be a lower-case letter. Lines are marked with the **k** command, described below.

/*RE*/ An *RE* is a Regular Expression, described under **Regular Expressions** below. When enclosed by slashes, *RE* addresses the first line found by searching for a matching string. The search proceeds forward from the line following the current line, and wraps through the beginning of the buffer to include all preceding lines, as well as the current line.

?*RE*? An *RE* enclosed in question marks addresses the first line containing a match found by searching backward from the line preceding the current line. The search wraps through the end of the buffer to include all lines following the current line (in reverse order), as well as the current line.

*address*±*n*

An address followed by a plus sign (+) or a minus sign (–), followed by a decimal number, specifies that address plus or minus the indicated number of lines. (The plus sign may be omitted.) If the address is omitted, the current line is used as the base. For example, '**31–3**' addresses line 28 in the buffer.

*address*±

If an address ends with '+' or '–', then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and the previous rule, the address '–' refers to the line preceding the current line. (To maintain compatibility with earlier versions of **ed**, the character '^' is equivalent to '–'.) Trailing '+' and '–' characters have a cumulative effect, so '––' refers to the current line, less 2.

, By itself, a comma stands for the address pair '**1,$**'.

; A semicolon by itself stands for the pair '**.,$**'.

By default for a given command. If you do not specify an address for a command to operate on, a command that requires an address supplies one by default. This is typically the current line.

A pair of addresses separated by a comma signifies an inclusive range of lines, and the current line is not changed unless the command changes it. When addresses are separated by a semicolon, however, the current line is set to the address preceding the semicolon before any subsequent addresses are interpreted. This feature can be used to determine the starting line for forward and backward searches using '/', and '?'.

The second address of any two-address sequence must correspond to a line that occurs later in the buffer than that of the first address.

**Regular Expressions**

**ed** supports a limited form of regular-expression notation, which can be used in a line address to specify lines by content. A regular expression (RE) specifies a set of character strings to match against — such as "any string containing digits 5 through 9" or "only lines containing uppercase letters." A member of this set of strings is said to be *matched* by the regular expression. Regular expressions or *patterns* are used to address lines in the buffer (see **Addresses** , above), and also for selecting strings to be replaced using the **s** (substitute) command.

Where multiple matches are present in a line, a regular expression matches the *longest* of the *leftmost* matching strings.

Regular expressions can be built up from the following "single-character" RE's:

*c* Any ordinary character not listed below. An ordinary character matches itself.

\ Backslash. When followed by a special character, the RE matches the "quoted" character. A backslash followed by one of <, >, (, ), {, or }, represents an *operator* in a regular expression, as described below.

. Dot. Matches any single character except NEWLINE.

^ As the leftmost character, a caret (or circumflex) constrains the RE to match the leftmost portion of a line. A match of this type is called an "anchored match" because it is "anchored" to a specific place in the line. The ^ character loses its special meaning if it appears in any position other than the start of the RE.

$ As the rightmost character, a dollar sign constrains the RE to match the rightmost portion of a line. The $ character loses its special meaning if it appears in any position other than at the end of the RE.

^*RE*$ The construction ^*RE* $ constrains the RE to match the entire line.

\< The sequence \< in an RE constrains the one-character RE immediately following it only to match something at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\> The sequence \> in an RE constrains the one-character RE immediately following it only to match something at the end of a "word."

[*c*...] A nonempty string of characters, enclosed in square brackets matches any single character in the string. For example, **[abcxyz]** matches any single character from the set '**abcxyz**'. When the first character of the string is a caret (^), then the RE matches any character *except* NEWLINE and those in the remainder of the string. For example, '**[^45678]**' matches any character except '**45678**'. A caret in any other position is interpreted as an ordinary character.

[]*c*...] The right square bracket does not terminate the enclosed string if it is the first character (after an initial '^', if any), in the bracketed string. In this position it is treated as an ordinary character.

[*l*–*r*] The minus sign, between two characters, indicates a range of consecutive ASCII characters to match. For example, the range '**[0–9]**' is equivalent to the string '**[0123456789]**'. Such a bracketed string of characters is known as a *character class*. The '–' is treated as an ordinary character if it occurs first (or first after an initial ^) or last in the string.

*d* Delimiter character. The character used to delimit an RE within a command is special for that command (for example, see how / is used in the **g** command, below).

The following rules and special characters allow for constructing RE's from single-character RE's:

A concatenation of RE's matches a concatenation of text strings, each of which is a match for a successive RE in the search pattern.

* A single-character RE, followed by an asterisk (*) matches *zero* or more occurrences of the single-character RE. Such a pattern is called a *closure*. For example, **[a–z][a–z]*** matches any string of one or more lower case letters.

\{m\}
\{m,\}
\{m,n\} A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is an RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be nonnegative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between m* and *n*, inclusively. Whenever a choice exists, the RE matches as many occurrences as possible.

\(...\) An RE enclosed between the character sequences \( and \) matches whatever the unadorned RE matches, but saves the string matched by the enclosed RE in a numbered substring register. There can be up to nine such substrings in an RE, and parenthesis operators can be nested.

*n* Match the contents of the *n*th substring register from the current RE. This provides a mechanism for extracting matched substrings. For example, the expression ^\(.*\)\1$ matches a line consisting entirely of two adjacent non-null appearances of the same string. When nested parenthesized substrings are present, *n* is determined by counting occurrences of \( starting from the left.

// The null RE (//) is equivalent to the last RE encountered.

**Commands**

The commands **a** for *append*, **c** for *change*, and **i** for *insert*, allow you to add new text to the buffer. While accepting new text, **ed** is said to be in *input mode*. While in input mode, *no* commands are recognized; all character input is inserted into the buffer. To exit from input mode, enter a dot (**.**) on a line by itself; **ed** then reverts to command mode. Or, you can interrupt **ed** (typically with CTRL-C), in which case it displays a **?** and returns to command mode.

Commands may accept zero, one, or two addresses. Commands that accept no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when too few addresses are given; if more addresses are given than such a command requires, only the last ones are used.

In the following list of **ed** commands, the default addresses are shown in parentheses; the parenthesized addresses are *not* part of the command.

It is generally illegal for more than one command to appear on a line. However, any command (except **e**, **f**, **r**, or **w**) may be followed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively.

(.)**a**
*text*
.

Append text. Add lines of *text* into the buffer after the addressed line. The resulting current line is the last line of input, or the addressed line if no text is entered. Address **0** is legal for this command, in which case the *text* is placed at the beginning of the buffer. The maximum number of characters per input line (from a terminal) is 256, including the final NEWLINE.

(.)**c**
*text*
.

Change lines. Delete the addressed lines, and then accept lines of *text* to replace them. **c** accepts one or two addresses; the default is the current line. The resulting current line is the last line of input, or the line preceding the deleted lines if no text is entered.

(.,.)**d** Delete the addressed lines from the buffer. **d** accepts one or two addresses; the default is the current line. The resulting current line is the line following the last one deleted; if the deleted lines were at the end of the buffer, the new last line is the resulting current line.

**e** *filename*

Edit a file. Delete the entire contents of the buffer, and then read in the named file. The resulting current line is the last line of the buffer. **e** reports the number of characters read into the buffer, and sets *filename* to be the current file (for use as a default filename in subsequent commands). If no *filename* is given, the current filename, if any, is used (see the **f** command, below). If *filename* is replaced by a shell (**sh**(1)) command prefaced with a '**!**', the shell command is executed and its output is read into the buffer after the current line. Such a shell command is *not* used as the current filename. **e** displays a **?** if the buffer has not been written out since the last change made — another **e** command in response to the **?** forces the command to take effect.

**E** *filename*

The **E** command is like **e**, except that the editor does not check for changes to the buffer since the last **w** command was performed.

**f** *filename*

Display or set the current filename. If *filename* is given as an argument, the file (**f**) command changes the current filename to *filename*; otherwise, it prints the current filename.

(**1**,**$**) **g**/*RE*/*command-list*

The global (**g**) command performs *command-list* on all lines in the range of addresses that match *RE*. **ed** executes *command-list* for each matching line in succession, setting the current line to each in turn. *command-list* can contain a single command, or it can be continued across input lines, with one **ed** command per line, by escaping all but the last NEWLINE with a \ character. Operations that place **ed** into input mode (**a**, **i**, and **c**), *are* permitted in *command-list*; the final '**.**'

terminating text input may be omitted if it is the last line of the *command-list*. **g**, **G**, **v**, and **V** commands, however, are *not* permitted. An empty *command-list* is equivalent to the **p** command.

(1,$) **G***/RE/*

The interactive **G** (Global) command, selects all lines that match the given *RE*. Then, each selected line is made current, and any *one* command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) can be performed upon that line. A NEWLINE acts as a null command; an **&** reexecutes the most recent command. Commands entered during execution of the **G** command can address and affect lines other than the current line. The **G** command can be terminated by an interrupt (typically CTRL-D).

**h** Help. Display a short error message that explains the reason for the most recent **?** diagnostic.

**H** Automatic printing of help diagnostics. Toggle between printing the **?** diagnostic, or automatically printing diagnostic messages as well.

(.)**i**
*text*
.

Insert Text. Insert the given *text* into the buffer, above the addressed line. **i** accepts one *address*; the default is the current line. The resulting current line is the last line of input; if no text is input, it is the line just before the addressed line. This command differs from the *a* command only in the placement of the input text; Address 0 is not allowed for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the NEWLINE character).

(.,.+1)**j** Join Lines. Remove the NEWLINE character from between the two addressed lines. The defaults are the current line and the line following. If exactly one address is given, this command does nothing. The joined line is the resulting current line.

(.)**k***c* Mark the addressed line with the name *c*, a lower-case letter. The address-form, *'c*, addresses the line marked by *c*. **k** accepts one *address*; the default is the current line. The current line is left unchanged.

(.,.)**l** List nonprinting characters. Print the addressed lines in an unambiguous way: a few nonprinting characters, such as TAB and BACKSPACE are represented by visually mnemonic overstrikes. All other nonprinting characters are shown in octal, with long lines folded. **l** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. An **l** command may be appended to any command other than **e**, **f**, **r**, or **w**.

(.,.) **m***address*

Move addressed lines to just after *address*. Address 0 is legal, and moves the addressed line(s) to the beginning of the file. An error results if *address* falls within the range of lines to move. **m** accepts two addresses to specify a range of lines to move; the default is the current line. The resulting current line is the last of the moved lines.

(.,.)**n** Number the displayed lines. Print the addressed lines, preceding each with its line number and a TAB character. **n** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. The **n** command can be appended to any command other than **e**, **f**, **r**, or **w**.

(.,.)**p** Print the addressed lines. **p** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. The **p** command may be appended to any command other than **e**, **f**, **r**, or **w**. For example, **dp** deletes the current line and prints the new current line.

**P** Toggle prompting on or off. When prompting is in effect, the editor prompts with a * for commands. A subsequent **P** command turns prompting off.

**q** Quit. Exit from **ed**. Note, however, that the buffer is *not* automatically written out; you must write any changes to be saved with the **w** command; **ed** warns you once if you have not saved your changes (unless the '–' option is in effect). A second **q** forces **ed** to exit regardless, destroying the buffer's contents.

**Q** Force quit. This is the same as **q**, but you do not get any warning if you have not previously written out the buffer. **ed** simply exits.

(**$**) **r** *filename*

Read in the contents of *filename*, after the addressed line. If *filename* is not given, the current filename, if any, is used (see the **e** and **f** commands). The current filename is *not* altered; if there is no current filename, *filename* becomes the current filename. **r** accepts one *address*; the default is **$**. Address 0 is legal for **r**, in which case the file is read in at the beginning of the buffer. If the read is successful, the number of characters read is typed. The resulting current line is the last line read in from the file. If *filename* is replaced by a shell (**sh**(1)) command prefaced with a **!**, the shell command is executed and its output is read in. Such a shell command is *not* remembered as the current filename.

(.,.) **s**/*RE*/*rs*/
(.,.) **s**/*RE*/*rs*/**g**
(.,.) **s**/*RE*/*rs*/*n*

Substitute. Search each addressed line for the first occurrence of a string matching the specified *RE*, and replace it with *rs*, the replacement string. If **g** (global suffix) is appended to the command, replace *all* (non-overlapped) matching strings in each addressed line with the replacement string *rs*. Note: the **g** *suffix* is distinct from the **g** *command*. If a number *n* is appended, replace only the *n*'th occurrence of the matched string on each addressed line. **s** accepts one or two addresses; the default is the current line. The resulting current line is the last line on which a substitution is made. An error results if *RE* matches no strings in the addressed line or range. Any character (other than SPACE or NEWLINE can be used instead of / to delimit *RE* and *rs*. As with RE's in addresses, you can refer to the entire string matched by *RE* with an '**&**'; you can refer to parenthesized substrings within *RE* using **\1** ... *****n*. When **%** is the only character in *rs*, the *rs* from in the most recent substitute command is used as the current *rs*. The **%** loses its special meaning when it is in a replacement string of more than one character, or if it is preceded by a backslash.

A line may be split by substituting a NEWLINE character into it. The NEWLINE in the *replacement* must be escaped by preceding with an '\'. Such substitutions cannot be done as part of a **g** or **v** command list.

(.,.) **t***address*

Transfer. Transpose a copy of the addressed range of lines to just after the given *address*. **t** (transfer) is like **m** (move), except that it copies of the lines, rather than moving them. **t** accepts two addresses preceding the operation letter, the current address is the default. The resulting current line is the last line copied. Address 0 is allowed.

**u** Undo. Reverse the effect of the most recent command that modified the buffer. A second **u** undoes the undo operation.

(**1**,**$**) **v**/*RE*/*command-list*

This command is the same as the global command **g** except that the *command-list* is executed with '.' initially set to every line that does *not* match the RE.

(**1**,**$**) **V**/*RE*

Similar to the **G** command, except that the lines selected are those that do *not* match the *RE*.

(**1**,**$**) **w** *filename*

Write the addressed lines to *filename*. If the file does not exist, **ed** creates it. The current filename is *not* altered; if there is no current filename, then *filename* becomes current. If no *filename* is given, the current filename, if any, is used. **w** accepts one or two addresses; the default is all lines in the file. The current line is unchanged. If the command is successful, the number of characters written is displayed. If *filename* is replaced by a shell (**sh**(1)) command prefaced with a '**!**', the shell command is executed with standard input taken from the addressed lines. Such a shell command is *not* remembered as the current filename.

(1,$) **W** *filename*
Like **w**, but append the addressed lines onto the named file.

**x** Encrypt the file. **ed** prompts for an encryption key from the standard input. Subsequent **e**, **r**, and **w** commands encrypt and decrypt the text with this key (see **crypt**(1)). An empty key turns off encryption. Encryption can also be specified on the command line with the **–x** option.

($)= Display the line number of the addressed line; the current line remains unchanged.

**!***shell-command*
Run a shell command. *shell-command* is a (Bourne shell) command line. **ed** replaces the unescaped character **%** with the current filename; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. (**!!** repeats the last shell command.) If any such expansion is performed, the expanded line is echoed. The current line is unchanged.

*address*
NEWLINE
An address, alone on a line, prints the addressed line. A NEWLINE alone is equivalent to '**.+1p**', which is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, **ed** prints a **?** and returns to *its* command level.

**File Format Specification Support**
**ed** supports the **fspec**(5) formatting capability for displaying lines. When the first line of a file is a format specification of the form:

**<:***ts*[**,***ts*]... *smax***:>**

where *ts* is the column number of a TAB stop and *max* is the maximum line length for display purposes, and with the terminal in '**stty –tabs**' or '**stty tab3**' mode (see **stty**(1V) for details), the indicated TAB stops are used in displayed lines. While inserting text, however, TAB stops are set to every eighth column.

## ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **ed**. On entry to **ed**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default.** When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## FILES

| | |
|---|---|
| **/usr/tmp/e#** | temporary; # is the process number |
| **ed.hup** | file for saved work if the terminal is hung up |

## SEE ALSO

**crypt**(1), **ex**(1), **grep**(1V), **sed**(1V), **sh**(1), **stty**(1V), **vi(1), regexp(3), fspec**(5), **locale**(5), **iso_8859_1**(7)

*Editing Text Files*

## LIMITATIONS

The following limitations apply:

512 characters per line.
256 characters per global command-list.
1024 characters per filename.
The limit on the number of lines depends on the amount of user memory:
each line takes 1 word.

When reading a file, **ed** discards ASCII NUL characters and all characters after the last NEWLINE. Files (such as executable images) that contain characters not in the ASCII set (bit 8 on) cannot be edited using **ed**.

If a file is not terminated by a NEWLINE character, **ed** adds one and prints a message saying that it has done so.

If the closing delimiter of an RE or of a replacement string (such as /) would be the last character before a NEWLINE, that delimiter can be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

| | |
|---|---|
| **s/s1/s2** | **s/s1/s2/p** |
| **g/s1** | **g/s1/p** |
| **?s1** | **?s1?** |

**DIAGNOSTICS**

**?** For command errors.

***?file:error***

For an inaccessible file (use the **h** (help) and **H** (Help) commands for detailed explanations).

If changes have been made in the buffer since the last **w** command, **ed** issues a warning **?** when a command is given that would destroy the buffers contents. A second **e** or **q** command at this point will take effect. The '–' and –**s** command-line options inhibit this feature.

**WARNINGS**

A **!** command cannot be subject to a **g** or a **v** command.

The sequence **\n** in an RE does not match a NEWLINE character.

Files encrypted directly with the **crypt**(1) command with the null key cannot be edited.

The encryption facilities of **ed** are not available on software shipped outside the U.S.

If the editor input is coming from a command file, the editor exits at the first failure of a command in that file.

## NAME

eject – eject media device from drive

## SYNOPSIS

**eject** [ **–dfnq** ] [ *device* | *nickname* ]

## AVAILABILITY

This command is not available on Sun 386*i* systems.

## DESCRIPTION

**eject** is used for those removable media devices that do not have a manual eject button. The device may be specified by its name or by a nickname; if no device is specified the default device is used.

Only devices that support **eject** under program control respond to this command.

It is not recommended to physically eject media from a device which contains mounted filesystems. **eject** automatically searches for any mounted filesystems which reside on the device and attempts to **umount** them prior to ejecting the media (see **mount**(8)). If the unmount operation fails, **eject** prints a warning message and exits. The **–f** flag may be used to specify an eject *even* if the device contains mounted partitions.

**eject** can also display its default device and a list of nicknames.

## OPTIONS

| | |
|---|---|
| **–d** | Display the name of the default device to be ejected. |
| **–f** | Force the device to eject even if it is busy. |
| **–n** | Display the nickname to device name translation table. |
| **–q** | Query to see if the media is present. |
| *device* | Specify which device to **eject**, by the name it appears in the directory **/dev**. |
| *nickname* | Specify which device to **eject**, by its nickname as known to this command. |

## FILES

| | |
|---|---|
| **/dev/rfd0c** | |
| **/dev/rsr0** | raw files |
| **/dev/sr0** | block files |

## SEE ALSO

**fd**(4S), **sr**(4S), **mount**(8)

## EXIT STATUS

**eject** returns the following exit codes:

| | |
|---|---|
| 0 | If the operation was successful or, with the **–q** option, the media *is* in the drive. |
| 1 | If the operation was unsuccessful or, with the **–q** option, the media is *not* in the drive. |
| 2 | If invalid flags were specified. |
| 3 | If an **ioctl**( ) request failed. |

## DIAGNOSTICS

A short help message is printed if an unknown flag is specified. A diagnostic is printed if the device name cannot be opened or does not support **eject**.

**Device Busy** An attempt was made to eject a device that has a mounted filesystem. A warning message is printed when doing a forced eject of a mounted device.

## BUGS

There should be a way to change the default on a per-user basis.

**NAME**

enablenumlock, disablenumlock – enable or disable the numlock key

**SYNOPSIS**

**enablenumlock**

**disablenumlock**

**AVAILABILITY**

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

**DESCRIPTION**

When you press the "Num Lock" key and then press one of the keys numbered R1 through R15 on the right keypad, the character on the upper part of the key is generated rather than the key's function.

In previous releases, "Num Lock" was enabled only in DOS windows and in some third-party UNIX applications. Currently, "Num Lock" is enabled by default for the right keypad in all SunView windows.

**enablenumlock** re-enables "Num Lock" for applications that expect it to be enabled.

**SEE ALSO**

*Sun386i User's Guide*

**BUGS**

The default enabling of the "Num Lock" key is incompatible with some applications with special work around code to enable the "Num Lock" key. These applications require the use of **disablenumlock**. DOS applications are not affected.

**NAME**

env – obtain or alter environment variables for command execution

**SYNOPSIS**

**env** [ – ] [ *name=value ...* ] [ *command* ]

**DESCRIPTION**

**env** obtains the current **environment**, modifies it according to its arguments, and executes the command with the modified environment that results.

If no *command* is specified, the resulting environment is displayed.

**OPTIONS**

| | |
|---|---|
| – | Ignore the environment that would otherwise be inherited from the current shell. Restricts the environment for *command* to that specified by the arguments. |
| *name=value* | Set the environment variable *filename* to *value* and add it to the environment. |

**SEE ALSO**

**sh**(1), **execve**(2V), **profil**(2), **environ**(5V)

## NAME

eqn, neqn, checkeq – typeset mathematics

## SYNOPSIS

**eqn** [ **–d***xy* ] [ **–f***n* ] [ **–p***n* ] [ **–s***n* ] [ *filename* ] ...

**neqn** [ *filename* ] ...

**checkeq** [ *filename* ] ...

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**eqn** (and **neqn**) are language processors to assist in describing equations. **eqn** is a preprocessor for **troff**(1) and is intended for devices that can print **troff**'s output. **neqn** is a preprocessor for **nroff**(1) and is intended for use with terminals. Usage is almost always:

**example%** **eqn** *filename* ... | **troff**
**example%** **neqn** *filename* ... | **nroff**

If no *filename*s are specified, **eqn** and **neqn** read from the standard input. A line beginning with **.EQ** marks the start of an equation; the end of an equation is marked by a line beginning with **.EN**. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as "delimiters"; subsequent text between delimiters is also treated as **eqn** input.

**checkeq** reports missing or unbalanced delimiters and **.EQ**/**.EN** pairs.

## OPTIONS

**–d***xy* Set equation delimiters set to characters *x* and *y* with the command-line argument. The more common way to do this is with **delim***xy* between **.EQ** and **.EN**. The left and right delimiters may be identical. Delimiters are turned off by **delim off** appearing in the text. All text that is neither between delimiters nor between **.EQ** and **.EN** is passed through untouched.

**–f***n* Change font to *n* globally in the document. The font can also be changed globally in the body of the document by using the **gfont** directive.

**–p***n* Reduce subscripts and superscripts by *n* point sizes from the previous size. In the absence of the **–p** option, subscripts and superscripts are reduced by 3 point sizes from the previous size.

**–s***n* Change point size to *n* globally in the document. The point size can also be changed globally in the body of the document by using the **gsize** directive.

## EQN LANGUAGE

Tokens within **eqn** are separated by braces, double quotes, tildes, circumflexes, SPACE, TAB, or NEWLINE characters. Braces { } are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full SPACE in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus '**x sub i**' makes $x_i$ , '**a sub i sup 2**' produces $a_i^2$, and '**e sup {x sup 2 + y sup 2}**' gives $e^{x^2+y^2}$.

Fractions are made with **over**: '**a over b**' yields $\frac{a}{b}$.

**sqrt** makes square roots: '**1 over sqrt {ax sup 2 +bx+c}**' results in $\frac{1}{\sqrt{ax^2+bx+c}}$ .

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim_{n \to \infty} \sum_{0}^{n} x_i$ is made with '**lim from {n–> inf } sum from 0 to n x sub i**'.

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: '**left [ x sup 2 + y sup 2 over alpha right ] ~=~1**' produces $\left[x^2+\frac{y^2}{\alpha}\right] = 1$. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: '**pile {a above b above c}**' produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**: '**matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }**' produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: '**x dot = f(t) bar**' is $\dot{x} = \overline{f(t)}$, '**y dotdot bar ~=~ n under**' is $\ddot{\bar{y}} = \underline{n}$, and '**x vec ~=~ y dyad**' is $\vec{x} = \vec{\vec{y}}$.

Sizes and font can be changed with **size** *n* or **size** ±*n*, **roman**, **italic**, **bold**, and **font** *n*. Size and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **–s***n* and **–f***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

**define** *thing % replacement %*

defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The *%* may be any character that does not occur in *replacement*.

Keywords like **sum** (∑), **int** (∫), **inf** (∞), and shorthands like **>=** (≥), **->** (→), and **!=** (≠) are recognized. Greek letters are spelled out in the desired case, as in **alpha** or **GAMMA**. Mathematical words like sin, cos, and log are made Roman automatically. **troff**(1) four-character escapes like \(bu (•) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with **troff** when all else fails.

**SEE ALSO**

**tbl**(1), **troff**(1), **eqnchar**(7), **ms**(7)

*Formatting Documents*

**BUGS**

To embolden digits, parens, etc., it is necessary to quote them, as in '**bold "12.3"**'.

## NAME

error – categorize compiler error messages, insert at responsible source file lines

## SYNOPSIS

**error** [ **–n** ] [ **–s** ] [ **–q** ] [ **–v** ] [ **–t** *suffixlist* ] [ **–I** *ignorefile* ] [ *filename* ]

## DESCRIPTION

**error** analyzes error messages produced by a number of compilers and language processors. It replaces the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously.

**error** looks at error messages, either from the specified file *filename* or from the standard input, and:

- Determines which language processor produced each error message.
- Determines the file name and line number of the erroneous line.
- Inserts the error message into the source file immediately preceding the erroneous line.

Error messages that can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. **error** touches source files only after all input has been read.

**error** is intended to be run with its standard input connected with a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into **error.** For example, when using the **csh** syntax,

**tutorial% make –s lint | & error**

will analyze all the error messages produced by whatever programs **make**(1) runs when making lint.

**error** knows about the error messages produced by: **make**(1), **cc**(1V), **cpp**(1), **as**(1), **ld**(1), **lint**(1V), and other compilers. For all languages except Pascal, error messages are restricted to one line. Some error messages refer to more than one line in more than one file, in which case **error** duplicates the error message and inserts it in all the appropriate places.

## OPTIONS

**–n** Do *not* touch any files; all error messages are sent to the standard output.

**–q** **error** asks whether the file should be touched. A '**y**' or '**n**' to the question is necessary to continue. Absence of the **–q** option implies that all referenced files (except those referring to discarded error messages) are to be touched.

**–v** After all files have been touched, overlay the visual editor **vi** with it set up to edit all files touched, and positioned in the first touched file at the first error. If **vi**(1) can't be found, try **ex**(1) or **ed**(1) from standard places.

**–t** *suffixlist*

Take the following argument as a suffix list. Files whose suffices do not appear in the suffix list are not touched. The suffix list is dot separated, and '*' wildcards work. Thus the suffix list:

**.c.y.f*.h**

allows **error** to touch files ending with '**.c**', '**.y**', '**.f***' and '**.h**'.

**–s** Print out statistics regarding the error categorization.

**error** catches interrupt and terminate signals, and terminates in an orderly fashion.

## USAGE

### Action Statements

**error** does one of six things with error messages.

**synchronize** Some language processors produce short errors describing which file they are processing. **Error** uses these to determine the file name for languages that don't include the file name in each error message. These synchronization messages are consumed entirely by **error.**

**discard** Error messages from **lint** that refer to one of the two **lint** libraries, **/usr/lib/lint/llib-lc** and **/usr/lib/lint/llib-port** are discarded, to prevent accidentaly touching these libraries. Again, these error messages are consumed entirely by **error.**

**nullify** Error messages from **lint** can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named **.errorrc** in the user's home directory, or from the file named by the **–I** option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

**not file specific** Error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They are not inserted into any source file.

**file specific** Error messages that refer to a specific file but to no specific line are written to the standard output when that file is touched.

**true errors** Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are inserted into source files. Other error messages are consumed entirely by **error** or are written to the standard output. **error** inserts the error messages into the source file on the line preceding the line number in the error message. Each error message is turned into a one line comment for the language, and is internally flagged with the string **###** at the beginning of the error, and **%%%** at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed. In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can wreak havoc with a future compilation. To avoid this, format the source program so there are no language statements on the same line as the end of a comment.

## FILES

| | |
|---|---|
| **~/.errorrc** | function names to ignore for **lint** error messages |
| **/dev/tty** | user's teletype |

## SEE ALSO

**as**(1), **cc**(1V), **cpp**(1), **csh**(1), **ed**(1), **ex**(1), **ld**(1), **lint**(1V), **make**(1), **vi**(1)

## BUGS

Opens the tty-device directly for user input.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's error message format may cause **error** to not understand the error message.

**error,** since it is purely mechanical, will not filter out subsequent errors caused by "floodgating" initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected, error puts them before. The alignment of the | marking the point of error is also disturbed by **error.**

**error** was designed for work on CRT 's at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

## NAME

ex, edit, e – line editor

## SYNOPSIS

**ex** [ – ] [ **–lLrRsvVxC** ] [ **–t** *tag* ] [ **+c** *command* | **–c** *command* ] *filename*...

**edit** [ *options* ]

## DESCRIPTION

**ex**, a line editor, is the root of a family of editors that includes **edit**, **ex**(1), and **vi**(1) (the display editor). In most cases **vi** is preferred for interactive use.

## OPTIONS

**– | –s** Suppress all interactive feedback to the user (useful for processing **ex** scripts in shell files).

**–l** Set up for editing LISP programs.

**–L** List the names of all files saved as the result of an editor or system crash.

**–r** Recover the indicated *filename*s after a system crash.

**–R** Read only. Do not overwrite the original file.

**–v** Start up in display editing state using **vi**. You can achieve the same effect by simply typing the **vi** command itself.

**–V** Verbose. Any non-tty input will be echoed on standard error. This may be useful when processing editor commands within shell scripts.

**–x** Prompt for a key to be used in encrypting the file being edited. When used in conjunction with a pre-existing file, *ex* will make an educated guess to determine whether or not the input text file is already encrypted.

**–C** Encryption option; the same as the **–x** option, except that all input text is assumed to have already been encrypted. This guarantees decryption in the cases where the **–x** option incorrectly determines that the input file is not already encrypted (this is extremely rare, and will only occur in conjunction with the use of files containing non-ASCII text).

**–t** *tag* Edit the file containing the tag *tag*. A tags database must first be created using the **ctags**(1) command.

**+c** *command*

**–c** *command*

Start the editing session by executing *command*.

## ENVIRONMENT

The editor recognizes the environment variable **EXINIT** as a command (or list of commands separated by | characters) to run when it starts up. If this variable is undefined, the editor checks for startup commands in the file **$HOME/.exrc** file, which you must own. However, if there is a **.exrc** owned by you in the current directory, the editor takes its startup commands from this file — overriding both the file in your home directory and the environment variable.

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **ex**. On entry to **ex**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default.** When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## FILES

| | |
|---|---|
| **/usr/lib/ex?.?strings** | error messages |
| **/usr/lib/ex?.?recover** | recover command |
| **/usr/lib/ex?.?preserve** | preserve command |

| | |
|---|---|
| **/etc/termcap** | describes capabilities of terminals |
| **.exrc** | editor startup file for current directory |
| **$HOME/.exrc** | user's editor startup file if **./.exrc** is not found |
| **/tmp/Ex***nnnnn* | editor temporary file |
| **/tmp/Rx***nnnnn* | file named buffer temporary |
| **/var/preserve** | preservation directory |

**SEE ALSO**

**awk**(1), **ctags**(1), **ed**(1), **grep**(1V), **sed**(1V), **vi**(1), **locale**(5), **termcap**(5), **environ**(5V), **iso_8859_1**(7)

*Editing Text Files*

**BUGS**

The **z** command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line '–' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

With the **modeline** option in effect, the editor checks the first five lines of the text file for commands of the form

**ex:** *command*:

or

**vi:** *command*:

if any are found, the editor executes them. This can result in unexpected behavior, and is not recommended in any case. In earlier releases, **modeline** was in effect by default. Now it is not, but setting it in the **.exrc** file or the **EXINIT** environment variable can still produce untoward effects.

**RESTRICTIONS**

The encryption facilities of **ex** are not available on software shipped outside the U.S.

## NAME

expand, unexpand – expand TAB characters to SPACE characters, and vice versa

## SYNOPSIS

**expand** [ *–tabstop* ] [ *–tab1, tab2,..., tabn* ] [ *filename ...* ]

**unexpand** [ **–a** ] [ *filename ...* ]

## DESCRIPTION

**expand** copies *filenames* (or the standard input) to the standard output, with TAB characters expanded to SPACE characters. BACKSPACE characters are preserved into the output and decrement the column count for TAB calculations. **expand** is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain TAB characters.

**unexpand** copies *filenames* (or the standard input) to the standard output, putting TAB characters back into the data. By default, only leading SPACE and TAB characters are converted to strings of tabs, but this can be overridden by the **–a** option (see the OPTIONS section below).

## OPTIONS

### expand

*–tabstop*

Specify as a single argument, sets TAB characters *tabstop* SPACE characters apart instead of the default 8.

*–tab1, tab2,..., tabn*

Set TAB characters at the columns specified by *tab1...*

### unexpand

**–a** Insert TAB characters when replacing a run of two or more SPACE characters would produce a smaller output file.

## NAME

expr – evaluate arguments as a logical, arithmetic, or string expression

## SYNOPSIS

**/bin/expr** *argument...*

## SYSTEM V SYNOPSIS

**/usr/5bin/expr** *argument...*

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**expr** evaluates expressions as specified by its arguments. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument, so terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note: **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, two's-complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by '\'. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

*expr* \| *expr*
: Return the first *expr* if it is neither NULL nor **0**, otherwise returns the second *expr*.

*expr* \& *expr*
: Return the first *expr* if neither *expr* is NULL or **0**, otherwise returns **0**.

*expr* { =, \>, \>= , \<, \<=, != } *expr*
: Return the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

*expr* { +, − } *expr*
: Addition or subtraction of integer-valued arguments.

*expr* { *, /, % } *expr*
: Multiplication, division, or remainder of the integer-valued arguments.

*string* : *regular-expression*
**match** *string regular-expression*
: The two forms of the matching operator above are synonymous. The matching operators : and **match** compare the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of **ed**(1), except that all patterns are "anchored" (treated as if they begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

**substr** *string integer-1 integer-2*
: Extract the substring of *string* starting at position *integer-1* and of length *integer-2* characters. If *integer-1* has a value greater than the length of *string*, **expr** returns a null string. If you try to extract more characters than there are in *string*, **expr** returns all the remaining characters from *string*. Beware of using negative values for either *integer-1* or *integer-2* as **expr** tends to run forever in these cases.

**index** *string character-list*
: Report the first position in *string* at which any one of the characters in *character-list* matches a character in *string*.

**length** *string*
Return the length (that is, the number of characters) of *string*.

**( expr )** Parentheses may be used for grouping.

**SYSTEM V DESCRIPTION**

The operators **substr**, **index**, and **length** are not supported.

**EXAMPLES**

1. **a=‘expr $a + 1‘**

   Adds 1 to the shell variable **a**.

2. **# ’For $a equal to either "/usr/abc/file" or just "file"’**
   **expr $a : ’.*/\(.*\)’ \| $a**

   Returns the last segment of a path name (that is, the filename part). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. **# A better representation of example 2.**
   **expr //$a : ’.*/\(.*\)’**

   The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. **expr $VAR : ’.*’**

   Returns the number of characters in **$VAR**.

**SEE ALSO**

**ed**(1), **sh**(1), **test**(1V)

**EXIT CODE**

**expr** returns the following exit codes:

| | |
|---|---|
| 0 | if the expression is neither null nor **0** |
| 1 | if the expression *is* null or **0** |
| 2 | for invalid expressions. |

**DIAGNOSTICS**

| | |
|---|---|
| **syntax error** | for operator/operand errors |
| **non-numeric argument** | if arithmetic is attempted on such a string |
| **division by zero** | if an attempt to divide by zero is made |

**BUGS**

After argument processing by the shell, **expr** cannot tell the difference between an operator and an operand except by the value. If **$a** is an =, the command:

**expr $a = ’=’**

looks like:

**expr = = =**

as the arguments are passed to **expr** (and they will all be taken as the = operator). The following works:

**expr X$a = X=**

Note: the **match**, **substr**, **length**, and **index** operators cannot themselves be used as ordinary strings. That is, the expression:

**example% expr index expurgatorious length**
**syntax error**
**example%**

generates the ‘**syntax error**’ message as shown instead of the value **1** as you might expect.

## NAME

fdformat – format diskettes for use with SunOS

## SYNOPSIS

**Sun386i Systems**

**fdformat** [ **–L** ] [ **–2** ]

**SPARCstation 1 and Sun-3/80 Systems**

**fdformat** [ **–eflv** ] [ *device* ]

## AVAILABLITY

This command is available on Sun386i, SPARCstation 1 and Sun-3/80 systems only.

## DESCRIPTION

**fdformat** is a utility for formatting floppy diskettes. All new blank diskettes must be formatted before they can be used. **fdformat** formats and verifies each track on the diskette, and terminates if it finds any bad sectors. All existing data on the diskette, if any, is destroyed by formatting.

By default, **fdformat** formats a high density diskette with a capacity of 1.44 megabytes. Use the **–L** or **–l** option to format low density diskettes (720K capacity). Note: it is not possible to put a low density format onto a high density floppy diskette. High density diskettes can be recognized by the high density detect hole in the lower right corner of the diskette, opposite the write protect hole in the lower left corner.

The SPARCstation1 and Sun-3/80 version of **fdformat** writes a SunOS label on the diskette in logical block 0 after it has been formatted. The label is required on SPARCstation 1 and Sun-3/80 systems if you intend to put a UNIX file system on the floppy diskette.

On Sun386i systems, use the **–2** option to format diskettes in the optional external 5 1/4" floppy drive.

To format a diskette for use under MS-DOS, use the MS-DOS FORMAT command in a DOS window on the Sun386i system.

## OPTIONS

**Sun386i Systems**

**–L** Format a low density diskette.

**–2** Format a diskette in the optional external 5.25-inch floppy drive.

**SPARCstation 1 and Sun-3/80 Systems**

**–e** Eject the diskette when done.

**–f** Force. Do not ask for confirmation before starting format.

**–l** Format a low density (720K) diskette.

**–v** Verify the floppy diskette after formatting.

## FILES

**Sun386i Systems**

| | |
|---|---|
| **/dev/rfd0c** | 1.44 megabyte 3.5-inch high density diskette drive |
| **/dev/rfdl0c** | 720 kilobyte 3.5-inch low density diskette drive |
| **/dev/rfd2c** | High density 5.25-inch floppy drive |
| **/dev/rfdl2c** | Low density 5.25-inch floppy drive |

**SPARCstation 1 and Sun-3/80 Systems**

**/dev/rfd0c**

## SEE ALSO

**dos**(1), **fd**(4S)

**BUGS**

Currently bad sector mapping is not supported on floppy diskettes. Therefore, a diskette is unusable if **fdformat** finds an error (bad sector).

## NAME

file – determine the type of a file by examining its contents

## SYNOPSIS

**file** [ **–f** *ffile* ] [ **–cL** ] [ **–m** *mfile* ] *filename*...

## DESCRIPTION

**file** performs a series of tests on each *filename* in an attempt to determine what it contains. If the contents of a file appear to be ASCII text, **file** examines the first 512 bytes and tries to guess its language.

**file** uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type.

## OPTIONS

**–c** Check for format errors in the magic number file. For reasons of efficiency, this validation is not normally carried out. No file type-checking is done under **–c**.

**–f** *ffile* Get a list of filenames to identify from *ffile*.

**–L** If a file is a symbolic link, test the file the link references rather than the link itself.

**–m** *mfile*
Use *mfile* as the name of an alternate magic number file.

## EXAMPLE

This example illustrates the use of **file** on all the files in a specific user's directory:

```
example% pwd
/usr/blort/misc
example% file *
code:           mc68020 demand paged executable
code.c:         c program text
counts:         ascii text
doc:            roff, nroff, or eqn input text
empty.file:     empty
libz:           archive random library
memos:          directory
project:        symbolic link to /usr/project
script:         executable shell script
titles:         ascii text
s5.stuff:       cpio archive
example%
```

## ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **file**. On entry to **file**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## FILES

**/etc/magic**

**SEE ALSO**

**locale**(5), **magic**(5)

**BUGS**

**file** often makes mistakes. In particular, it often suggests that command files are C programs.

Does not recognize Pascal or LISP.

## NAME

find – find files by name, or by other characteristics

## SYNOPSIS

**find** *pathname-list expression*
**find** *component*

## DESCRIPTION

**find** recursively descends the directory hierarchy for each pathname in the *pathname-list*, seeking files that match a logical *expression* written using the operators listed below.

**find** does *not* follow symbolic links to other files or directories; it applies the selection criteria to the symbolic links themselves, as if they were ordinary files (see **ln**(1V) for a description of symbolic links).

If the *fast-find* feature is enabled, **find** displays pathnames in which a filename *component* occurs.

## USAGE

### Operators

In the descriptions, the argument *n* is used as a decimal integer where *+n* means more than *n*, *−n* means less than *n*, and *n* means exactly *n*.

**−fstype** *type* — True if the filesystem to which the file belongs is of type *type*, where *type* is typically **4.2** or **nfs**.

**−name** *filename* — True if the *filename* argument matches the current file name. Shell argument syntax can be used if escaped (watch out for **[**, **?** and ***).

**−perm** *onum* — True if the file permission flags exactly match the octal number *onum* (see **chmod**(1V)). If *onum* is prefixed by a minus sign, more flag bits (017777, see **chmod**(1V)) become significant and the flags are compared: (*flags*&*onum*)==*onum*.

**−prune** — Always yields true. Has the side effect of pruning the search tree at the file. That is, if the current path name is a directory, **find** will not descend into that directory.

**−type** *c* — True if the type of the file is *c*, where *c* is one of:

| | |
|---|---|
| **b** | for block special file **c** |
| **c** | for character special file |
| **d** | for directory |
| **f** | for plain file |
| **p** | for named pipe (FIFO) |
| **l** | for symbolic link |
| **s** | for socket |

**−links** *n* — True if the file has *n* links.

**−user** *uname* — True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** database, it is taken as a user ID.

**−nouser** — True if the file belongs to a user *not* in the **/etc/passwd** database.

**−group** *gname* — True if the file belongs to group *gname*. If *gname* is numeric and does not appear as a login name in the **/etc/group** database, it is taken as a group ID.

**−nogroup** — True if the file belongs to a group *not* in the **/etc/group** database.

**−size** *n* — True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.

**−inum** *n* — True if the file has inode number *n*.

**−atime** *n* — True if the file has been accessed in *n* days.
Note: the access time of directories in *path-name-list* is changed by **find** itself.

**−mtime** *n* — True if the file has been modified in *n* days.

**–ctime** *n* True if the file has been changed in *n* days. "Changed" means either that the file has been modified or some attribute of the file (its owner, its group, the number of links to it, etc.) has been changed.

**–exec** *command* True if the executed *command* returns a zero value as exit status. The end of *command* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current pathname.

**–ok** *command* Like **–exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

**–print** Always true; the current pathname is printed.

**–ls** Always true; prints current pathname together with its associated statistics. These include (respectively) inode number, size in kilobytes (1024 bytes), protection mode, number of hard links, user, group, size in bytes, and modification time. If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by '–>'. The format is identical to that of **ls –gilds** (see **ls**(1V)).
Note: formatting is done internally, without executing the **ls** program.

**–cpio** *device* Always true; write the current file on *device* in **cpio**(5) format (5120-byte records).

**–ncpio** *device* Always true; write the current file on *device* in **cpio –c** format (5120-byte records).

**–newer** *file* True if the current file has been modified more recently than the argument *filename*.

**–xdev** Always true; find does *not* traverse down into a file system different from the one on which current *argument* pathname resides.

**–depth** Always true; **find** descends the directory hierarchy, acting on the entries in a directory before acting on the directory itself. This can be useful when **find** is used with **cpio**(1) to transfer files that are contained in directories without write permission.

(*expression*) True if the parenthesized *expression* is true.
Note: Parentheses are special to the shell and must be escaped.

**!***primary* True if the *primary* is false (**!** is the unary *not* operator).

*primary1* [ **–a** ] *primary2*
True if both *primary1* and *primary2* are true. The **–a** is not required. It is implied by the juxtaposition of two primaries.

*primary1* **–o** *primary2*
True if either *primary1* or *primary2* is true (**–o** is the *or* operator).

**Fast-Find**

The fast-find feature is enabled by the presence of the **find.codes** database in **/usr/lib/find**. You must be **root** to build or update this database by running the **updatedb** script in that same directory. You may wish to modify the **updatedb** script to suit your needs.

An alternate database can be specified by setting the **FCODES** environment variable.

**EXAMPLE**

In our local development system, we keep a file called TIMESTAMP in all the manual page directories. Here is how to find all entries that have been updated since TIMESTAMP was created:

```
example% find /usr/share/man/man2 –newer /usr/share/man/man2/TIMESTAMP –print
/usr/share/man/man2
/usr/share/man/man2/socket.2
/usr/share/man/man2/mmap.2
example%
```

To find all the files called **intro.ms** starting from the current directory:

```
example% find . -name intro.ms -print
./manuals/assembler/intro.ms
./manuals/sun.core/intro.ms
./manuals/driver.tut/intro.ms
./manuals/sys.manager/uucp.impl/intro.ms
./supplements/general.works/unix.introduction/intro.ms
./supplements/programming.tools/sccs/intro.ms
example%
```

To recursively print all files names in the current directory and below, but skipping SCCS directories:

```
example% find . -name SCCS -prune -o -print
example%
```

To recursively print all files names in the current directory and below, skipping the contents of SCCS directories, but printing out the SCCS directory name:

```
example% find . -print -name SCCS -prune
example%
```

To remove files beneath your home directory named **a.out** or ***.o** that have not been accessed for a week and that are not mounted using NFS:

```
example% cd
example% find . \( -name a.out -o -name '*.o' \) -atime +7 -exec rm { } \; -o -fstype nfs -prune
```

**FILES**

| | |
|---|---|
| **/usr/lib/find/find.codes** | database for fast find |
| **/usr/lib/find/updatedb** | script to update fast-find database |
| **/usr/lib/find/code** | fast-find database utilities |
| **/usr/lib/find/bigram** | |

**ENVIRONMENT**

| | |
|---|---|
| **FCODES** | alternate database for fast find |

**SEE ALSO**

**chmod**(1V), **cpio**(1), **ln**(1V), **ls**(1V), **sh**(1), **test**(1V), **cpio**(5), **fs**(5)

## NAME

finger – display information about users

## SYNOPSIS

**finger** [ *options* ] *name*...

## DESCRIPTION

By default, **finger** displays information about each logged-in user, including his or her: login name, full name, terminal name (prepended with a '*' if write-permission is denied), idle time, login time, and location (comment field in **/etc/ttytab** for users logged in locally, hostname for users logged in remotely) if known.

Idle time is minutes if it is a single integer, hours and minutes if a ': ' is present, or days and hours if a **d** is present.

When one or more *name* arguments are given, more detailed information is given for each *name* specified, whether they are logged in or not. A *name* may be a first or last name, or an account name. Information is presented in a multi-line format, and includes, in addition to the information mentioned above:

the user's home directory and login shell
the time they logged in if they are currently logged in, or the time they last logged in if they are not, as well as the terminal or host from which they logged in and, if a terminal, the comment field in **/etc/ttytab** for that terminal
the last time they received mail, and the last time they read their mail
any plan contained in the file **.plan** in the user's home directory
and any project on which they are working described in the file **.project** (also in that directory)

If a *name* argument contains an at-sign, '@', then a connection is attempted to the machine named after the at-sign, and the remote finger daemon is queried. The data returned by that daemon is printed. If a long format printout is to be produced, **finger** passes the **–l** option to the remote finger daemon over the network using the **/W** feature of the protocol (see *NAME/FINGER Protocol)*.

## OPTIONS

**–m** Match arguments only on user name (not first or last name).

**–l** Force long output format.

**–s** Force short output format.

**–q** Force quick output format, which is similar to short format except that only the login name, terminal, and login time are printed.

**–i** Force "idle" output format, which is similar to short format except that only the login name, terminal, login time, and idle time are printed.

**–b** Suppress printing the user's home directory and shell in a long format printout.

**–f** Suppress printing the header that is normally printed in a non-long format printout.

**–w** Suppress printing the full name in a short format printout.

**–h** Suppress printing of the **.project** file in a long format printout.

**–p** Suppress printing of the **.plan** file in a long format printout.

## FILES

| | |
|---|---|
| **/etc/utmp** | who is logged in |
| **/etc/passwd** | for users' names |
| **/var/adm/lastlog** | last login times |
| **/etc/ttytab** | terminal locations |
| **~/.plan** | plans |
| **~/.project** | projects |

**SEE ALSO**

**passwd**(1), **w**(1), **who**(1), **whois**(1)

Harrenstien, K., *NAME/FINGER Protocol*, 1977 December 30, RFC 742.

**BUGS**

Only the first line of the **.project** file is printed.

## NAME

fmt, fmt_mail – simple text and mail-message formatters

## SYNOPSIS

**fmt** [ **–cs** ] [ *–width* ] [ *inputfile*... ]

**fmt_mail** [ **–cs** ] [ *–width* ] [ *inputfile* ... ]

## DESCRIPTION

**fmt** is a simple text formatter that fills and joins lines to produce output lines of (up to) the number of characters specified in the *–width* option. The default *width* is 72. **fmt** concatenates the *inputfile*s listed as arguments. If none are given, **fmt** formats text from the standard input.

Blank lines are preserved in the output, as is the spacing between words. **fmt** does not fill lines beginning with '.', for compatibility with **nroff**(1). Nor does it fill lines starting with 'From:' (but for full compatibility with **mail**(1), use **fmt_mail**).

Indentation is preserved in the output, and input lines with differing indentation are not joined (unless **–c** is used).

**fmt** can also be used as an in-line text filter for **vi**(1); the **vi** command:

**!}fmt**

reformats the text between the cursor location and the end of the paragraph.

**fmt_mail** is a script that formats and sends mail messages. It leaves mail header lines untouched, and runs the remainder of the message through **fmt -s**. The resulting message is passed along to **sendmail**(8), which routes it to the recipient.

## OPTIONS

**–c** Crown margin mode. Preserve the indentation of the first two lines within a paragraph, and align the left margin of each subsequent line with that of the second line. This is useful for tagged paragraphs.

**–s** Split lines only. Do not join short lines to form longer ones. This prevents sample lines of code, and other such "formatted" text, from being unduly combined.

*–width* Fill output lines to up to *width* columns.

## ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **fmt**. On entry to **fmt**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## SEE ALSO

**mail**(1), **nroff**(1), **vi**(1)

**NAME**

fold – fold long lines for display on an output device of a given width

**SYNOPSIS**

**fold** [ *–width* ] [ file ]

**DESCRIPTION**

Fold the contents of the specified *files*, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using **expand**(1) before using *fold*.

**SEE ALSO**

**expand**(1)

**BUGS**

Folding may not work correctly if underlining is present.

## NAME

fontedit – a vfont screen-font editor

## SYNOPSIS

**fontedit** [ *generic-tool-argument* ] ... [ *font_name* ]

## AVAILABILITY

This command is available with the *SunView User's* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**fontedit** is an editor for fixed-width fonts in *vfont* format whose characters are no taller than 24 pixels (larger characters will not fit completely onto the screen). For a description of *vfont* format, see **vfont**(5).

## OPTIONS

*generic-tool-argument*

**fontedit** accepts any generic tool argument as described in **sunview**(1). Otherwise, you can manipulate the tool using the Frame Menu.

## COMMANDS

To edit a font, type '**fontedit**'. A *font_name* may be supplied on the command line or may be typed into the Control panel once the program has started. If it exists, the *font_name* file must be in *vfont* format. When the program starts, it displays a single large window containing four subwindows. From top to bottom, the four subwindows are:

1) The top subwindow, a message subwindow, displays messages, prompts, and warnings.

2) The second subwindow from the top, an Control panel, allows you to set global parameters for the entire font and specify operations for editing any single character. The options are:

**(Load)** Load in the font specified in the file name field. The program will warn you if you try to read over a modified font.

**(Store)** Store the current font onto disk with the name in file name field.

**(Quit)** Quit the program; warns you if you have modified the font.

**Font name:**
The name of the font.

**Max Width** and **Max Height:**
The size, in pixels, of the largest character in the font. If you edit an existing font, these parameters are set automatically; you must set them if you are creating a new font. Changing either of these values for an existing font may alter the glyph of some characters of the font. If the glyph size of a character is larger than the new max size, then that character is clipped to the new size (its bottom and right edges are moved in). However, if a glyph's size is smaller than the new size, the glyph is left alone.

**Caps Height** and **X-Height:**
The distance, in pixels, between the top of a capital and lowercase letter and the baseline. When an existing font is edited, the values of **Caps Height** and **X-Height** are estimated by *fontedit*, and may require some adjustment.

**Baseline:** The number of pixels from the top (that is, the upper left corner) of the character to the baseline. For an existing font, the value of the largest baseline distance is used. For a new font, each character will have the same baseline distance. If this value is changed, then the baseline distance for all characters in the font will be the new value.

**(Apply)** Apply the current values of **Max Width**, **Max Height**, **Caps Height**, **X-Height**, and **Baseline** to the font. That is, changes made to these values do not take effect until **Apply** is selected.

**Operation:**

This is a list of drawing and editing operations that you can perform on a character. For drawing, the left mouse button draws in black, and the middle draws in white. Operations are:

**Single Pt** Press a mouse button down and a grey cell will appear; move the mouse and the cell will follow it. Releasing the button will draw.

**Pt Wipe** Pressing a button down will draw and moving with the button down will continue drawing until the button is released.

**Line** Button down marks the end point of a line; moving with the button down rubber bands a line; releasing button draws the line.

**Rect** Like **Line** except draws a rectangle.

**Cut** Button down marks one end of rectangle, and moving rubber bands the outline of the rectangle. Button up places the contents of the rectangle into a buffer and then "cuts" (draws in white) the rectangular region from the character. The **Paste** operation (below) gets the data from the buffer.

**Copy** Like **Cut** except that the region is just copied; no change is made to the character.

**Paste** Button down displays a rectangle the size of the region in the buffer. Moving with the button down moves the rectangle. Button up pastes the contents of the buffer into the character.
The contents of the **paste** buffer cannot be transferred between tools.
In **Copy** or **Cut** mode, holding down the shift key while pressing the left or middle mouse button will perform a **Paste** action. For best results, after placing a region in the buffer, press down the shift key and hold it down, then press down the mouse button. Release the mouse key to paste the region and then release the shift key.

3) The third subwindow echoes the characters in the current font as they are typed. Note that the cursor must be in this window in order to see the characters. Your character delete key will delete the echoed characters.

4) The bottom subwindow, the editing subwindow, displays eight smaller squares at its top; these are called **edit buttons.** The top section of each of these buttons contains a line of text in the form *nnn: c*, where *nnn* is the hexadecimal number of the character and *c* is the standard ASCII character corresponding to that number. In the lower section of the button the character of the current font, if it exists, is displayed. Clicking once over an editing button selects its character for editing.

   Just below this row of buttons is a box with the characters "0 9 A Z a z" in it. This box is called a **slider.** The slider allows you to scroll around in the font and select which section of the font you want displayed in the edit buttons. The black rectangle near "a" is an indicator which shows the section of the font that is displayed in the buttons above. To move the indicator, select it by pressing the left or middle mouse button down over the indicator and then move the mouse to the left or right with the button down; the indicator will slide along with the cursor. Releasing the button selects the new section of the font. A faster method of moving about in the font is to just press down and release the mouse button above the area you want without bothering to drag the indicator. Another method of scrolling through the characters of the font is to press a key on the keyboard when the cursor is in the bottom window; that character is the first one displayed in the edit buttons.

**EDITING CHARACTERS:**

To edit a character, click once over the edit button where the character is displayed. When you do this, an edit pad will appear in the bottom subwindow.

The edit pad consists of an editing area bordered by scales, a proof area, and 3 command buttons. The editing area is **Max Width** by **Max Height** when the pad opens, and displays a magnified view of the selected character. Black squares indicate foreground pixels. The editing area is surrounded by scales which show the current **Caps Height**, **X-Height** and **Baseline** in reverse video.

Just outside the scales, on the top, right side, and bottom of the pad, are three small boxes with the capital letters "R", "B", and "A" in them. These boxes are movable sliders that change the right edge, bottom edge, and x-axis advance of the character respectively. In a fixed-width font, these values are usually the same for all characters; however, in a variable-width font these controls can be used to set these properties for each character.

To the right of the pad is the proof area where the character is displayed at normal (that is, screen) resolution and three buttons. The three buttons are:

**Undo** Clicking the left or middle mouse button undoes the last operation.

**Store** Stores the current representation of the character in the font.

**Quit** Closes the edit pad.

In the bottom subwindow, the right mouse button displays a menu of operations. These operations are the same as those in the control panel discussed above; you can select the current operation by either picking the operation in the control panel or by selecting the appropriate menu with the right button of the mouse. When the cursor is in the other subwindows, the right button displays the standard tool menu.

## ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **fontedit**. On entry to **fontedit**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## FILES

**/usr/lib/fonts/fixedwidthfonts**
Sun-supplied screen fonts

## SEE ALSO

**sunview**(1), **vswap**(1), **locale**(5), **vfont**(5), **iso_8859_1**(7)

## BUGS

Results are unpredictable with variable-width fonts. The baseline should be greater than 0 or else the font cannot be read in by **fontedit** or by **sunview**(1).

**NAME**

foption – determine available floating-point code generation options

**SYNOPSIS**

**foption** [ –f*type* ]

**AVAILABILITY**

This command is not available for the Sun386i.

**DESCRIPTION**

**foption** has two uses. Called without an argument, it sends a string to standard output which is the compiler floating-point option corresponding to the type of floating-point hardware that would be used by a program compiled with **–fswitch**. Exit status is undefined. This usage is intended for interactively determining available floating-point hardware. On systems without floating-point hardware, the result would be

```
example% foption
soft
```

corresponding to the compiler option **–fsoft**.

Called with an argument which is one of the compiler floating-point options **–ffpa**, **–f68881**, **–fsoft**, or **–fswitch**, it produces no output but returns exit status 0 (true) if a program compiled with that option could execute on this machine, and status 1 (false) otherwise. Thus **foption –fsoft** and **foption –fswitch** always produce exit status 0. This usage is intended for shell scripts and Makefiles that, for instance, select different executable files or link with different libraries according to the floating-point hardware present.

**foption** is undefined on Sun-4 systems since there are no floating-point code generation options.

**OPTIONS**

**–f***type* Return exit status 0 if a program compiled **–f***type* could execute on this machine.

**SEE ALSO**

**cc**(1V), **fpaversion**(8), **mc68881version**(8)

**NAME**

from – display the sender and date of newly-arrived mail messages

**SYNOPSIS**

**from** [ **–s** *sender* ] [ *username* ]

**DESCRIPTION**

**from** prints out the mail header lines in your mailbox file to show you who your mail is from. If *username* is specified, then *username*'s mailbox is examined instead of your own.

**OPTIONS**

**–s** *sender* Only display headers for mail sent by *sender*.

**FILES**

**/var/spool/mail/***

**SEE ALSO**

**biff**(1), **mail**(1), **old-prmail**(1)

## NAME

ftp – file transfer program

## SYNOPSIS

**ftp** [ **–dgintv** ] [ *hostname* ]

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**ftp** is the user interface to the ARPANET standard File Transfer Protocol (FTP). **ftp** transfers files to and from a remote network site.

The client host with which **ftp** is to communicate may be specified on the command line. If this is done, **ftp** immediately attempts to establish a connection to an FTP server on that host; otherwise, **ftp** enters its command interpreter and awaits instructions from the user. When **ftp** is awaiting commands from the user, it displays the prompt '**ftp>**'.

## OPTIONS

Options may be specified at the command line, or to the command interpreter.

**–d** Enable debugging.

**–g** Disable filename "globbing."

**–i** Turn off interactive prompting during multiple file transfers.

**–n** Do not attempt "auto-login" upon initial connection. If auto-login is enabled, **ftp** checks the **.netrc** file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, **ftp** will prompt for the login name of the account on the remote machine (the default is the login name on the local machine), and, if necessary, prompts for a password and an account with which to login.

**–t** Enable packet tracing (unimplemented).

**–v** Show all responses from the remote server, as well as report on data transfer statistics. This is turned on by default if **ftp** is running interactively with its input coming from the user's terminal.

## COMMANDS

**!** [ *command* ]

Run *command* as a shell command on the local machine. If no *command* is given, invoke an interactive shell.

**$** *macro-name* [ *args* ]

Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.

**account** [ *passwd* ]

Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

**append** *local-file* [ *remote-file* ]

Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being altered by any **ntrans** or **nmap** setting. File transfer uses the current settings for "representation type", "file structure", and "transfer mode".

**ascii** Set the "representation type" to "network ASCII". This is the default type.

**bell** Sound a bell after each file transfer command is completed.

**binary** Set the "representation type" to "image".

**bye** Terminate the FTP session with the remote server and exit **ftp**. An EOF will also terminate the session and exit.

**case** Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

**cd** *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

**cdup** Change the remote machine working directory to the parent of the current remote machine working directory.

**close** Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

**cr** Toggle RETURN stripping during "network ASCII" type file retrieval. Records are denoted by a RETURN/LINEFEED sequence during "network ASCII" type file transfer. When **cr** is on (the default), RETURN characters are stripped from this sequence to conform with the UNIX system single LINEFEED record delimiter. Records on non-UNIX-system remote hosts may contain single LINEFEED characters; when an "network ASCII" type transfer is made, these LINEFEED characters may be distinguished from a record delimiter only when **cr** is off.

**delete** *remote-file*
Delete the file *remote-file* on the remote machine.

**debug** [ *debug-value* ]
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level. When debugging is on, **ftp** prints each command sent to the remote machine, preceded by the string '**-->**'.

**dir** [ *remote-directory* ] [ *local-file* ]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is '–', output is sent to the terminal.

**disconnect**
A synonym for **close**.

**form** [ *format-name* ]
Set the carriage control format subtype of the "representation type" to *format-name*. The only valid *format-name* is **non-print**, which corresponds to the default "non-print" subtype.

**get** *remote-file* [ *local-file* ]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current **case**, **ntrans**, and **nmap** settings. The current settings for "representation type", "file structure", and "transfer mode" are used while transferring the file.

**glob** Toggle filename expansion, or "globbing", for **mdelete**, **mget** and **mput**. If globbing is turned off, filenames are taken literally.

Globbing for **mput** is done as in **csh**(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine, and the lists are not merged.

Expansion of a directory name is likely to be radically different from expansion of the name of an ordinary file: the exact result depends on the remote operating system and FTP server, and can be previewed by doing '**mls** *remote-files* –'.

**mget** and **mput** are not meant to transfer entire directory subtrees of files. You can do this by transferring a **tar**(1) archive of the subtree (using a "representation type" of "image" as set by the **binary** command).

**hash** Toggle hash-sign (#) printing for each data block transferred.

**help** [ *command* ]
Print an informative message about the meaning of *command*. If no argument is given, **ftp** prints a list of the known commands.

**lcd** [ *directory* ]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

**ls** [ *remote-directory* ] [ *local-file* ]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, or if *local-file* is '–', the output is sent to the terminal.

**macdef** *macro-name*
Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive NEWLINE characters in a file or RETURN characters from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed.

The macro processor interprets '$' and '\' as special characters. A '$' followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A '$' followed by an '**i**' signals that macro processor that the executing macro is to be looped. On the first pass '**$i**' is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A '\' followed by any character is replaced by that character. Use the '\' to prevent special treatment of the '$'.

**mdelete** [ *remote-files* ]
Delete the *remote-files* on the remote machine.

**mdir** *remote-files local-file*
Like **dir**, except multiple remote files may be specified. If interactive prompting is on, **ftp** will prompt the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

**mget** *remote-files*
Expand the *remote-files* on the remote machine and do a **get** for each file name thus produced. See **glob** for details on the filename expansion. Resulting file names will then be processed according to **case**, **ntrans**, and **nmap** settings. Files are transferred into the local working directory, which can be changed with '**lcd** *directory*'; new local directories can be created with '**! mkdir** *directory*'.

**mkdir** *directory-name*
Make a directory on the remote machine.

**mls** *remote-files local-file*
Like **ls**(1V), except multiple remote files may be specified. If interactive prompting is on, **ftp** will prompt the user to verify that the last argument is indeed the target local file for receiving **mls** output.

**mode** [ *mode-name* ]
Set the "transfer mode" to *mode-name*. The only valid *mode-name* is **stream**, which corresponds to the default "stream" mode.

**mput** *local-files*
Expand wild cards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting file names will then be processed according to **ntrans** and **nmap** settings.

**nmap** [ *inpattern outpattern* ]

Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during **mget** commands and **get** commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX-system remote host with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *inpattern* is a template for incoming filenames (which may have already been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences **$1**, **$2**, ..., **$9** in *inpattern*. Use \ to prevent this special treatment of the **$** character. All other characters are treated literally, and are used to determine the **nmap** *inpattern* variable values.

For example, given *inpattern* **$1.$2** and the remote file name **mydata.data**, **$1** would have the value "mydata", and **$2** would have the value "data".

The *outpattern* determines the resulting mapped filename. The sequences **$1**, **$2**, ..., **$9** are replaced by any value resulting from the *inpattern* template. The sequence **$0** is replaced by the original filename. Additionally, the sequence '[*seq1* ,*seq2* ]' is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*.

For example, the command '**nmap $1.$2.$3 [$1,$2].[$2,file]**' would yield the output filename **myfile.data** for input filenames **myfile.data** and **myfile.data.old**, **myfile.file** for the input filename **myfile**, and **myfile.myfile** for the input filename **.myfile**. SPACE characters may be included in *outpattern*, as in the example '**nmap $1 | sed "s/ *$//" > $1**'. Use the \ character to prevent special treatment of the '$', '[', ']' and ',' characters.

**ntrans** [ *inchars* [ *outchars* ] ]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote target filename, and characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX-system remote host with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

**open** *host* [ *port* ]

Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, **ftp** will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), **ftp** will also attempt to automatically log the user in to the FTP server (see below).

**prompt** Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. By default, prompting is turned on. If prompting is turned off, any **mget** or **mput** will transfer all files, and any **mdelete** will delete all files.

**proxy** *ftp-command*

Execute an FTP command on a secondary control connection. This command allows simultaneous connection to two remote FTP servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command '**proxy ?**' to see other FTP commands executable on the secondary connection.

The following commands behave differently when prefaced by **proxy**: **open** will not define new macros during the auto-login process, **close** will not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection.

Third party file transfers depend upon support of the **PASV** command by the server on the secondary control connection.

**put** *local-file* [ *remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any **ntrans** or **nmap** settings in naming the remote file. File transfer uses the current settings for "representation type", "file structure", and "transfer mode".

**pwd** Print the name of the current working directory on the remote machine.

**quit** A synonym for **bye**.

**quote** *arg1 arg2 ...*
Send the arguments specified, verbatim, to the remote FTP server. A single FTP reply code is expected in return.

**recv** *remote-file* [ *local-file*]
A synonym for **get**.

**remotehelp** [ *command-name* ]
Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

**rename** *from to*
Rename the file *from* on the remote machine to have the name *to*.

**reset** Clear reply queue. This command re-synchronizes command/reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the FTP protocol by the remote server.

**rmdir** *directory-name*
Delete a directory on the remote machine.

**runique**
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a **get** or **mget** command, a '**.1**' is appended to the name. If the resulting name matches another existing file, a '**.2**' is appended to the original name. If this process continues up to '**.99**', an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note: **runique** will not affect local files generated from a shell command (see below). The default value is off.

**send** *local-file* [ *remote-file* ]
A synonym for **put**.

**sendport**
Toggle the use of **PORT** commands. By default, **ftp** will attempt to use a **PORT** command when establishing a connection for each data transfer. The use of **PORT** commands can prevent delays when performing multiple file transfers. If the **PORT** command fails, **ftp** will use the default data port. When the use of **PORT** commands is disabled, no attempt will be made to use **PORT** commands for each data transfer. This is useful when connected to certain FTP implementations that ignore **PORT** commands but incorrectly indicate they have been accepted.

**status** Show the current status of **ftp**.

**struct** [ *struct-name* ]
Set the "file structure" to *struct-name*. The only valid *struct-name* is **file**, which corresponds to the default "file" structure.

**sunique**
Toggle storing of files on remote machine under unique file names. The remote FTP server must support the **STOU** command for successful completion. The remote server will report the unique name. Default value is off.

**tenex** Set the "representation type" to that needed to talk to TENEX machines.

**trace** Toggle packet tracing (unimplemented).

**type** [ *type-name* ]
Set the "representation type" to *type-name*. The valid *type-name*s are **ascii** for "network ASCII", **binary** or **image** for "image", and **tenex** for "local byte size" with a byte size of 8 (used to talk to TENEX machines). If no type is specified, the current type is printed. The default type is "network ASCII".

**user** *user-name* [ *password* ] [ *account* ]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, **ftp** will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless **ftp** is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

**verbose** Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose mode is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose mode is on if **ftp**'s commands are coming from a terminal, and off otherwise.

**?** [ *command* ]
A synonym for **help.**

Command arguments which have embedded spaces may be quoted with quote (") marks.

If any command argument which is not indicated as being optional is not specified, **ftp** will prompt for that argument.

**ABORTING A FILE TRANSFER**

To abort a file transfer, use the terminal interrupt key (usually CTRL-C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending a ftp protocol **ABOR** command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for **ABOR** processing. If the remote server does not support the **ABOR** command, an "ftp>" prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when **ftp** has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the **ABOR** processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local **ftp** program must be killed by hand.

**FILE NAMING CONVENTIONS**

Local files specified as arguments to **ftp** commands are processed according to the following rules.

1) If the file name '–' is specified, the standard input (for reading) or standard output (for writing) is used.

2) If the first character of the file name is '|', the remainder of the argument is interpreted as a shell command. **ftp** then forks a shell, using **popen**(3S) with the argument supplied, and reads (writes) from the standard output (standard input) of that shell. If the shell command includes SPACE characters, the argument must be quoted; for example '"| **ls** –**lt**"'. A particularly useful example of this mechanism is: '**dir | more**'.

3) Failing the above checks, if "globbing" is enabled, local file names are expanded according to the rules used in the **csh**(1); see the **glob** command. If the **ftp** command expects a single local file (for example, **put**), only the first filename generated by the "globbing" operation is used.

4) For **mget** commands and **get** commands with unspecified local file names, the local filename is the remote filename, which may be altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.

5) For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **sunique** is on.

**FILE TRANSFER PARAMETERS**

The FTP specification specifies many parameters which may affect a file transfer.

The "representation type" may be one of "network ASCII", "EBCDIC", "image", or "local byte size" with a specified byte size (for PDP-10's and PDP-20's mostly). The "network ASCII" and "EBCDIC" types have a further subtype which specifies whether vertical format control (NEWLINE characters, form feeds, etc.) are to be passed through ("non-print"), provided in TELNET format ("TELNET format controls"), or provided in ASA (FORTRAN) ("carriage control (ASA)") format. **ftp** supports the "network ASCII" (subtype "non-print" only) and "image" types, plus "local byte size" with a byte size of 8 for communicating with TENEX machines.

The "file structure" may be one of "file" (no record structure), "record", or "page". **ftp** supports only the default value, which is "file".

The "transfer mode" may be one of "stream", "block", or "compressed". **ftp** supports only the default value, which is "stream".

**SEE ALSO**

**csh**(1), **ls**(1V), **rcp**(1C), **tar**(1), **popen**(3S), **netrc**(5), **ftpd**(8C)

**BUGS**

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2 BSD code handling transfers with a "representation type" of "network ASCII" has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2 BSD servers using a "representation type" of "network ASCII". Avoid this problem by using the "image" type.

**NAME**

gcore – get core images of running processes

**SYNOPSIS**

**gcore** [ **–o** *filename* ] *process-id* ...

**DESCRIPTION**

**gcore** creates a core image of each specified process. Such an image can be used with **adb**(1) or **dbx**(1). The name of the core image file for the process whose process ID is *process-id* will be **core.***process-id*.

**OPTIONS**

**–o** *filename* Substitute *filename* in place of **core** as the first part of the name of the core image files.

**FILES**

**core.***process-id* core images

**SEE ALSO**

**adb**(1), **csh**(1), **dbx**(1), **kill**(1), **ptrace**(2)

**NAME**

getopt – parse command options in shell scripts

**SYNOPSIS**

**set -- 'getopt** *opstring* **$*'**
**set argv = ('getopt** *opstring* **$*')**

**AVAILABILITY**

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**getopt** breaks up options in command lines for easy parsing by shell scripts, and checks for legal options. *optstring* is a string of option letters to recognize, (see **getopt**(3)). If a letter is followed by a colon, the option is expected to have an argument — which may or may not be separated by white space.

(The '--' following **set** indicates that the Bourne shell is to pass arguments beginning with a dash as parameters to the script.)

If '–' appears on the command line that invokes the script, **getopt** uses it to delimit the end of options it is to parse (see example below). If used explicitly, **getopt** will recognize it; otherwise, **getopt** will generate it at the first argument it encounters that has no '–'. In either case, **getopt** places it at the end of the options. The positional parameters ($1 $2...) of the shell are reset so that each option in *optstring* is broken out and preceded by a '–', along with the argument (if any) for each.

**EXAMPLE**

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
#! /usr/bin/sh
set -- getopt abo: $*'
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b) FLAG =$i; shift;;
        -o)              OARG =$2; shift 2;;
        --)              shift; break;;
        esac
done
```

This code will accept any of the following command lines as equivalent:

```
cmd -a -o arg  f1  f2
cmd -aoarg  f1  f2
cmd -oarg -a  f1  f2
cmd -a -oarg --  f1  f2
```

**SEE ALSO**

**csh**(1), **getopts**(1), **sh**(1), **getopt**(3)

**DIAGNOSTICS**

**getopt** prints an error message on the standard error when it encounters an option letter not included in *optstring*.

**NOTES**

**getopts**(1) is preferred.

**NAME**

getopts, getoptcvt – parse command options in shell scripts

**SYNOPSIS**

**getopts** *optstring name* [ *argument...* ]

**getoptcvt [ –b ]** *filename*

**DESCRIPTION**

**getopts** is used by shell procedures to parse positional parameters and to check for legal options. It should be used in place of the **getopt**(1V) command. It supports the following command syntax rules:

- Option names must be one character long.
- All options must be preceded by '–'.
- Options with no arguments may be grouped after a single '–'.
- Option-arguments cannot be optional.
- All options must precede operands on the command line.
- '––' may be used to indicate the end of the options.

*optstring* must contain the option letters the command using **getopts** will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, **getopts** will place the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell procedure is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, **getopts** places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name*.

When the end of options is encountered, **getopts** exits with a non-zero exit status. The special option '––' may be used to delimit the end of the options.

By default, **getopts** parses the positional parameters. If extra arguments (*argument ...*) are given on the **getopts** command line, **getopts** will parse them instead.

**getoptcvt** reads the shell script in *filename*, converts it to use **getopts** instead of **getopt**, and writes the results on the standard output.

**OPTIONS**

**getoptcvt**

**–b** Generate a script that will be portable to earlier releases of the UNIX system. The script will determine at run time whether to invoke **getopts** or **getopt**.

**EXAMPLE**

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
        case $c in
        a | b)          FLAG=$c;;
        o)              OARG=$OPTARG;;
        \?)             echo $USAGE
                        exit 2;;
        esac
done
shift 'expr $OPTIND - 1'
```

This code will accept any of the following as equivalent:

**cmd** **–a** **–b** **–o** **"xxx z yy"** *filename*
**cmd** **–a** **–b** **–o** **"xxx z yy"** **—** *filename*
**cmd** **–ab** **–o** **xxx,z,yy** *filename*
**cmd** **–ab** **–o** **"xxx z yy"** *filename*
**cmd** **–o** **xxx,z,yy** **–b** **–a** *filename*

**SEE ALSO**

**getopt**(1V), **sh**(1), **getopt**(3)

**WARNING**

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

**DIAGNOSTICS**

**getopts** prints an error message on the standard error when it encounters an option letter not included in *optstring*.

**NAME**

get_selection – copy the contents of a SunView selection to the standard output

**SYNOPSIS**

**get_selection** [ *rank* ] [ **t** *seconds* ] [ **D** ]

**AVAILABILITY**

This command is available with the *SunView User's* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**get_selection** prints the contents of the indicated selection on standard out. A selection is a collection of objects (for instance, characters) picked with the mouse in the SunView window system.

**OPTIONS**

*rank* Indicate which selection is to be printed:
**1**: primary;
**2**: secondary;
**3**: clipboard.

The default is primary.

**t seconds** Indicate how many seconds to wait for the holder of a selection to respond to a request before giving up. The default is 6 seconds.

**D** Debugging. Inquire through a special debugging service for the selection, rather than accessing the standard service. Useful only for debugging window applications which are clients of the selection library.

**EXAMPLE**

The following line in a SunView root menu file provides a menu command to print the primary selection on the user's default printer:

```
"Print It"  sh -c get_selection | lpr
```

**SEE ALSO**

*SunView User's Guide*

**NAME**

gfxtool – run graphics programs in a SunView window

**SYNOPSIS**

**gfxtool** [ **–C** ] [ *program* [ *arguments* ] ]

**AVAILABILITY**

This command is available with the *SunView User's* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**gfxtool** is a standard tool provided with the *SunView* environment. It runs graphics programs that do not overwrite the terminal emulator from which they run.

**gfxtool** has two subwindows: a terminal subwindow and an empty subwindow. The terminal subwindow contains a running shell, just like the shelltool (see **shelltool**(1)). Programs invoked in the terminal subwindow can run in the empty subwindow. You can move the boundary between these two subwindows as described in **sunview**(1). If you wish, you can make **gfxtool** your console by entering –C as the first argument.

Normally you can use the mouse and keyboard anywhere in the empty subwindow to access frame functions. However, some graphics programs which run in this window may take over inputs directed to it. For example, SunCore uses the mouse and keyboard for its own input. When you run such tools, access the Frame Menu from the tool boundaries or frame header.

**gfxtool** also accepts all of the generic tool arguments; see **sunview**(1) for a list of these arguments.

If a *program* argument is present, **gfxtool** runs it. If there are no arguments, **gfxtool** runs the program corresponding to your **SHELL** environment variable. If this environment variable is not available, then **gfxtool** runs **sh**(1).

**OPTIONS**

**–C** Redirect system console output to this instance of **gfxtool**.

**FILES**

**~/.ttyswrc**
**/usr/demo/***

**SEE ALSO**

**sh**(1), **shelltool**(1), **sunview**(1), **graphics_demos**(6)

**BUGS**

If more than 256 characters are input to a terminal emulator subwindow without an intervening NEWLINE, the terminal emulator may hang. If this occurs, display the Frame Menu; the '**TTY Hung?**' submenu there has one item, '**Flush input**', that you can invoke to correct the problem.

## NAME

gprof – display call-graph profile data

## SYNOPSIS

**gprof** [ **–abcsz** ] [ **–e** *function-name* ] [ **–E** *function-name* ] [ **–f** *function-name* ] [ **–F** *function-name* ] [ *image-file* [ *profile-file* ... ] ]

## DESCRIPTION

**gprof** produces an execution profile of a program. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file which is created by programs compiled with the **–pg** option of **cc**(1V) and other compilers. That option also links in versions of the library routines which are compiled for profiling. The symbol table in the executable image file *image-file* (**a.out** by default) is read and correlated with the call graph profile file *profile-file* (**gmon.out** by default). If more than one profile file is specified, the **gprof** output shows the sum of the profile information in the given profile files.

First, execution times for each routines are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. The first listing shows the functions sorted according to the time they represent, including the time of their call graph descendants. Below each function entry is shown its (direct) call-graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendants is propagated to its (direct) call-graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

Next, a flat profile is given, similar to that provided by **prof**(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time. Finally, an index showing the correspondence between function names and call-graph profile index numbers.

A single function may be split into subfunctions for profiling by means of the **MARK** macro (see **prof**(3)).

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. It is assumed that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call-graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

The profiled program must call **exit**(2V) or return normally for the profiling information to be saved in the **gmon.out** file.

## OPTIONS

**–a** Suppress printing statically declared functions. If this option is given, all relevant information about the static function (for instance, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the **a.out** file.

**–b** Brief. Suppress descriptions of each field in the profile.

**–c** The static call-graph of the program is discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

**–s** Produce a profile file **gmon.sum** which represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of **gprof** (probably also with a **–s**) option to accumulate profile data across several runs of an **a.out** file.

**–z** Display routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the **–c** option for discovering which routines were never called.

**–e** *function-name*

Suppress printing the graph profile entry for routine *function-name* and all its descendants (unless they have other ancestors that are not suppressed). More than one **–e** option may be given. Only one *function-name* may be given with each **–e** option.

**–E** *function-name*

Suppress printing the graph profile entry for routine *function-name* (and its descendants) as **–e**, above, and also exclude the time spent in *function-name* (and its descendants) from the total and percentage time computations. More than one **–E** option may be given. For example:

'**–E** *mcount* **–E** *mcleanup*'

is the default.

**–f** *function-name*

Print the graph profile entry only for routine *function-name* and its descendants. More than one **–f** option may be given. Only one *function-name* may be given with each **–f** option.

**–F** *function-name*

Print the graph profile entry only for routine *function-name* and its descendants (as **–f**, above) and also use only the times of the printed routines in total time and percentage computations. More than one **–F** option may be given. Only one *function-name* may be given with each **–F** option. The **–F** option overrides the **–E** option.

## ENVIRONMENT

**PROFDIR**

If this environment variable contains a value, place profiling output within that directory, in a file named *pid.programname*. *pid* is the process ID, and *programname* is the name of the program being profiled, as determined by removing any path prefix from the **argv[0]** with which the program was called. If the variable contains a NULL value, no profiling output is produced. Otherwise, profiling output is placed in the file **gmon.out**.

## FILES

| | |
|---|---|
| **a.out** | executable file containing namelist |
| **gmon.out** | dynamic call-graph and profile |
| **gmon.sum** | summarized dynamic call-graph and profile |
| **$PROFDIR**/*pid.programname* | |

## SEE ALSO

**cc**(1V), **prof**(1), **tcov**(1), **exit**(2V), **profil**(2), **monitor**(3), **prof**(3)

Graham, S.L., Kessler, P.B., McKusick, M.K., '*gprof: A Call Graph Execution Profiler*', *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

## BUGS

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call-graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

## NAME

graph – draw a graph

## SYNOPSIS

**graph** [ **–a** *spacing* [ *start* ] ] [ **–b** ] [ **–c** *string* ] [ **–g** *gridstyle* ] [ **–l** *label* ]
[ **–m** *connectmode* ] [ **–s** ] [ **–x** [ **l** ] *lower* [ *upper* [ *spacing* ] ] ]
[ **–y** [ **l** ] *lower* [ *upper* [ *spacing* ] ] ] [ **–h** *fraction* ] [ **–w** *fraction* ] [ **–r** *fraction* ]
[ **–u** *fraction* ] [ **–t** ] ...

## DESCRIPTION

**graph** with no options takes pairs of numbers from the standard input as abscissaes and ordinates of a graph. Successive points are connected by straight lines. The standard output from **graph** contains plotting instructions (see **plot**(5)) suitable for input to **plot**(1G) or to the command **lpr –g** (see **lpr**(1)).

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes "...", in which case they may be empty or contain blanks and numbers; labels never contain NEWLINE characters.

A legend indicating grid range is produced with a grid unless the –s option is present.

## OPTIONS

Each option is recognized as a separate argument. If a specified lower limit exceeds the upper limit, the axis is reversed.

**–a** *spacing*[ *start* ] Supply abscissaes automatically (they are missing from the input); *spacing* is the spacing (default 1). *start* is the starting point for automatic abscissaes (default 0 or lower limit given by **–x**).

**–b** Break (disconnect) the graph after each label in the input.

**–c** *string* *String* is the default label for each point.

**–g** *gridstyle* *Gridstyle* is the grid style: 0 no grid, 1 frame with ticks, 2 full grid (default).

**–l** *label* is label for graph.

**–m** *connectmode* Mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.

**–s** Save screen, do not erase before plotting.

**–x** [ **l** ] *lower* [ *upper* [ *spacing* ] ]
If **l** is present, *x* axis is logarithmic. *lower* and *upper* are lower (and upper) *x* limits. *spacing*, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

**–y** [ **l** ] *lower* [ *upper* [ *spacing* ] ]
If **l** is present, *y* axis is logarithmic. *lower* and *upper* are lower (and upper) *y* limits. *spacing*, if present, is grid spacing on *y* axis. Normally these quantities are determined automatically.

**–h** *fraction* *fraction* of space for height.

**–w** *fraction* *fraction* of space for width.

**–r** *fraction* *fraction* of space to move right before plotting.

**–u** *fraction* *fraction* of space to move up before plotting.

**–t** Transpose horizontal and vertical axes. Option **–x** now applies to the vertical axis.

## SEE ALSO

**lpr**(1), **plot**(1G), **spline**(1G)

**BUGS**

**graph** stores all points internally and drops those for which there is no room.

Segments that run out of bounds are dropped, not windowed.

Logarithmic axes may not be reversed.

**NAME**

grep, egrep, fgrep – search a file for a string or regular expression

**SYNOPSIS**

**grep** [ **–bchilnsvw** ] [ **–e** *expression* ] [ *filename*... ]

**egrep** [ **–bchilnsv** ] [ **–e** *expression* ] [ **–f** *filename* ] [ *expression* ] [ *filename*... ]

**fgrep** [ **–bchilnsvx** ] [ **–e** *string* ] [ **–f** *filename* ] [ *string* ] [ *filename*... ]

**SYSTEM V SYNOPSIS**

**/usr/5bin/grep** [ **–bchilnsvw** ] [ **–e** *expression* ] [ *filename*... ]

**AVAILABILITY**

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

Commands of the **grep** family search the input *filenames* (the standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. **grep** patterns are limited regular expressions in the style of **ed**(1). **egrep** patterns are full regular expressions including alternation. **fgrep** patterns are fixed strings — no regular expression metacharacters are supported.

In general, **egrep** is the fastest of these programs.

Take care when using the characters '**$**', '*', [, '^', '|', '(', ')', and '\' in the *expression*, as these characters are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'.

When any of the **grep** utilities is applied to more than one input file, the name of the file is displayed preceding each line which matches the pattern. The filename is not displayed when processing a single file, so if you actually want the filename to appear, use **/dev/null** as a second file in the list.

**OPTIONS**

**–b** Precede each line by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

**–c** Display a count of matching lines rather than displaying the lines which match.

**–h** Do not display filenames.

**–i** Ignore the case of letters in making comparisons — that is, upper and lower case are considered identical.

**–l** List only the names of files with matching lines (once) separated by NEWLINE characters.

**–n** Precede each line by its relative line number in the file.

**–s** Work silently, that is, display nothing except error messages. This is useful for checking the error status.

**–v** Invert the search to only display lines that *do not* match.

**–w** Search for the expression as a word as if surrounded by \< and \>. This applies to **grep** only.

**–x** Display only those lines which match exactly — that is, only lines which match in their entirety. This applies to **fgrep** only.

**–e** *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a '–'.

**–e** *string*
For **fgrep** the argument is a literal character *string*.

**–f** *filename*
Take the regular expression (**egrep**) or a list of strings separated by NEWLINE (**fgrep**) from *filename*.

**SYSTEM V OPTIONS**

The **−s** option to **grep** indicates that error messages for nonexistent or unreadable files should be suppressed, not that all messages *except* for error messages should be suppressed.

**REGULAR EXPRESSIONS**

The following *one-character* regular expressions match a *single* character:

*c* An ordinary character (*not* one of the special characters discussed below) is a one-character regular expression that matches that character.

*c* A backslash (\) followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:

- '.', '*', '[', and '\' (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([ ]).
- '^' (caret or circumflex), which is special at the *beginning* of an *entire* regular expression, or when it immediately follows the left of a pair of square brackets ([ ]).
- $ (currency symbol), which is special at the *end* of an entire regular expression.

A backslash followed by one of '<', '>', '(', ')', '{', or '}', represents a special operator in the regular expression; see below.

. A '.' (period) is a one-character regular expression that matches any character except NEWLINE.

[ *string* ]

A non-empty string of characters enclosed in square brackets is a one-character regular expression that matches *any one* character in that string. If, however, the first character of the string is a '^' (a circumflex or caret), the one-character regular expression matches any character *except* NEWLINE and the remaining characters in the string. The '^' has this special meaning *only* if it occurs first in the string. The '−' (minus) may be used to indicate a range of consecutive ASCII characters; for example, **[0−9]** is equivalent to **[0123456789]**. The '−' loses this special meaning if it occurs first (after an initial '^', if any) or last in the string. The ']' (right square bracket) does not terminate such a string when it is the first character within it (after an initial '^', if any); that is, **[ ]a−f]** matches either ']' (a right square bracket ) or one of the letters **a** through **f** inclusive. The four characters '.', '*', '[ ', and '\' stand for themselves within such a string of characters.

The following rules may be used to construct regular expressions:

* A one-character regular expression followed by '*' (an asterisk) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

\(and\) A regular expression enclosed between the character sequences \( and \) matches whatever the unadorned regular expression matches. This applies only to **grep**.

*n* The expression *n* matches the same string of characters as was matched by an expression enclosed between \( and \) *earlier* in the same regular expression. Here *n* is a digit; the subexpression specified is that beginning with the *n*th occurrence of \( counting from the left. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string.

**Concatenation**

The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

\< The sequence \< in a regular expression constrains the one-character regular expression immediately following it only to match something at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\> The sequence \> in a regular expression constrains the one-character regular expression immediately following it only to match something at the end of a "word"; that is, either at the end of a line, or just before a character which is neither a letter, digit, nor underline.

\{*m*\}
\{*m*,\}
\{*m*,*n*\} A regular expression followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} matches a **range** of occurrences of the regular expression. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the regular expression matches as many occurrences as possible.

^ A circumflex or caret (^) at the beginning of an entire regular expression constrains that regular expression to match an *initial* segment of a line.

$ A currency symbol ($) at the end of an entire regular expression constrains that regular expression to match a *final* segment of a line.

The construction

**example%** ^*entire regular expression* $

constrains the entire regular expression to match the entire line.

**egrep** accepts regular expressions of the same sort **grep** does, except for \(, \), *n*, \<, \>, \{, and \}, with the addition of:

* A regular expression (not just a one-character regular expression) followed by '*' (an asterisk) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

\+ A regular expression followed by '+' (a plus sign) is a regular expression that matches *one* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

? A regular expression followed by '?' (a question mark) is a regular expression that matches *zero* or *one* occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

| Alternation: two regular expressions separated by '|' or NEWLINE match either a match for the first or a match for the second.

() A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence of operators at the same parenthesis level is '[ ]' (character classes), then '*' '+' '?' **(closures)**,then concatenation, then '|' **(alternation)**and NEWLINE.

**EXAMPLES**

Search a file for a fixed string using **fgrep**:

**example% fgrep intro /usr/share/man/man3/*.3***

Look for character classes using **grep**:

**example% grep '[1-8]([CJMSNX])' /usr/share/man/man1/*.1**

Look for alternative patterns using **egrep**:

**example% egrep '(Sally|Fred) (Smith|Jones|Parker)' telephone.list**

To get the filename displayed when only processing a single file, use **/dev/null** as the second file in the list:

```
example% grep 'Sally Parker' telephone.list /dev/null
```

**FILES**

**/dev/null**

**SEE ALSO**

**awk**(1), **ed**(1), **ex**(1), **sh**(1), **vi**(1), **sed**(1V)

**BUGS**

Lines are limited to 1024 characters by **grep**; longer lines are truncated.

The combination of **–l** and **–v** options does *not* produce a list of files in which a regular expression is not found. To get such a list, use the Bourne shell construct:

```
for filename in *
do
        if [ `grep "re" $filename | wc -l` -eq 0 ]
        then
                echo $filename
        fi
done
```

or the C shell construct:

```
foreach filename (*)
        if (`grep "re" $filename | wc -l` == 0) echo $filename
end
```

Ideally there should be only one **grep**.

**DIAGNOSTICS**

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

**NAME**

groups – display a user's group memberships

**SYNOPSIS**

**groups** [ *username* ... ]

**DESCRIPTION**

With no arguments, **groups** displays the groups to which you belong; else it displays the groups to which the *username* belongs. Each user belongs to a group specified in the password file **/etc/passwd** and possibly to other groups as specified in the file **/etc/group**. If you do not own a file but belong to the group which it is owned by then you are granted group access to the file.

**FILES**

**/etc/passwd**
**/etc/group**

**SEE ALSO**

**getgroups**(2V)

**NAME**

head – display first few lines of specified files

**SYNOPSIS**

**head** [ *–n* ] [ *filename*... ]

**DESCRIPTION**

**head** copies the first *n* lines of each *filename* to the standard output. If no *filename* is given, **head** copies lines from the standard input. The default value of *n* is 10 lines.

When more than one file is specified, the start of each file which looks like:

==>*filename*<==

Thus, a common way to display a set of short files, identifying each one, is:

**example% head –9999** *filename1 filename2* ...

**EXAMPLE**

The following example:

**example% head –4 /usr/share/man/man1/{cat,head,tail}.1***

produces:

```
==> /usr/share/man/man1/cat.1v <==
.TH CAT 1V "2 June 1983"
.SH NAME
cat – concatenate and display
.SH SYNOPSIS

==> /usr/share/man/man1/head.1 <==
.TH HEAD 1 "24 August 1983"
.SH NAME
head – display first few lines of specified files
.SH SYNOPSIS

==> /usr/share/man/man1/tail.1 <==
.TH TAIL 1  "27 April 1983"
.SH NAME
tail – display the last part of a file
.SH SYNOPSIS
```

**SEE ALSO**

**cat**(1V), **more**(1), **tail**(1)

## NAME

help_open – use help_viewer to open a file

## SYNOPSIS

**help_open** [ **–a** ] [ *filename* [ *page_number* ] ]

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

**help_open** is used to cause a running **help_viewer**(1) to open a file. *filename* is typically the name of a **help_viewer** file. A call is made to **help_viewer** using the same RPC mechanism as is used by Spot Help.

If *filename* is relative, **help_viewer** looks for it relative to the default help directory (as defined in the user's **.defaults** database). Otherwise, **help_viewer** treats *filename* as an absolute pathname.

If the RPC call to **help_viewer** fails, **help_open** attempts to spawn **help_viewer**, with *filename* as a command line argument. If the **–a** command line option was given, then *filename* is first converted to an absolute pathname, as described in OPTIONS, below.

## OPTIONS

**–a** Convert *filename* to an absolute pathname. This option causes **help_open** to get the current working directory and append it to the front of *filename* (thus creating an absolute pathname) before passing *filename* on to **help_viewer**. This allows **help_open** to be used with other processes, such as Sun Organizer (see **organizer**(1)), which deal in relative pathnames. The **–a** option has no effect if *filename* begins with the character '/'.

## EXAMPLES

In the first example, **help_viewer** opens the file **help/Help_Basics**. This file is located relative to the default help directory (as defined in the user's **.defaults** database). If the default help directory is set to **/vol/help/language/USA-English**, this is **/vol/help/language/USA-English/help/Help_Basics**.

```
example% help_open help/Help_Basics
example%
```

The second example is the same as the first, but it opens **Help_Basics** to page 3.

```
example% help_open "help/Help_Basics 3"
example%
```

In the next example, **help_viewer** opens **somefile** using the absolute pathname, **/home/mtravis/somefile**.

```
example% help_open /home/mtravis/somefile
example%
```

In the last example, **help_viewer** opens **/home/ahinkle/anotherfile**.

```
example% cd /home/ahinkle
example% help_open –a anotherfile
example%
```

## FILES

**/usr/lib/help/***

## SEE ALSO

**help_viewer**(1), **organizer**(1), **help**(5), **help_viewer**(5)

*Sun386i User's Guide*
*Sun386i Developer's Guide*

**NAME**

help_viewer – SunView application providing help with applications and desktop

**SYNOPSIS**

**/usr/lib/help_viewer** [ **–A** *time* ] [ **–dir** *dirname* ] [ *filename* [ # ] ]

**AVAILABILITY**

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

**DESCRIPTION**

**help_viewer** accesses help documentation about SunView applications and the SunView Desktop. This help consists of intermixed text and graphics This help consists of intermixed text and graphics displayed in a window called the Help Viewer.

You start and control **help_viewer** by one of these methods:

- Typing **help_viewer** at a shell prompt.
- Clicking on the More Help button in a Spot Help window.
- Sending instructions to the Help Viewer using the **help_open** command.

*filename* is the name of startup file relative to help directory (or **/vol/help** by default, as set in the Help category of defaultsedit). # is a page number separated from the filename by a SPACE. If # is omitted, the first page is shown.

The documentation within **help_viewer** is extensible, but as shipped it includes handbooks for the Desktop, **mailtool**(1), **shelltool**(1), **textedit**(1), **sunview**(1), **organizer**(1), **dos**(1), **coloredit**(1), **snap**(1), and itself (**help_viewer**).

Developers and users can include additional handbooks by modifying **/vol/help/format/Top_Level**. See **help_viewer**(5).

The user moves between the various pages of help with the assistance of hypertext *links*. Links are connections between pages of text. The convention is to use underlined text to indicate the presence of a link. When the user double-clicks on a link, the text associated with the topic indicated by the link is shown in the Help Viewer. There are links in many places to make it quick and easy to go from place to place within the **help_viewer** database.

Many help topics contain more than one page of text. In these cases, links to the next page and to the previous page are available at the upper-right corner of the Help Viewer, allowing the user to page through the document.

The user's current position within the hierarchy of text is indicated by the links at the upper-left corner of the Help Viewer. The last link in the list is the level just above the user's current position.

**OPTIONS**

The standard SunView options for window size, position, fonts, and other options are accepted. But note that the font setting only affects the font in the Help Viewer namestripe. Also, Help Viewer text does not wrap as a window is resized. See **sunview**(1) for details.

**–A** *time* Sets autoclose for *time* minutes. If the help window is not used for *time* minutes, it closes automatically.

**–dir** *dirname* Name of the help directory

**FILES**

**/vol/help** automount point of miscellaneous help files

The files in **/usr/lib/help** are used by the **help** and the **help_viewer** facilities, and the SCCS **sccs-help**(1) facility.

Directories within **/usr/lib/help** named after SunView applications and the Desktop contain specific information used by **help_viewer**. See **help_viewer**(5) for information about the files in these directories.

**SEE ALSO**

**sccs-help**(1), **mailtool**(1), **shelltool**(1), **textedit**(1), **help_open**(1), **help_viewer**(5)

*Sun386i User's Guide*

*Sun386i Developer's Guide*

**DIAGNOSTICS**

**help_viewer**(1) displays a pop-up error window if it cannot find the file required to show the requested help.

**NAME**

hostid – print the numeric identifier of the current host

**SYNOPSIS**

**hostid**

**DESCRIPTION**

The **hostid** command prints the identifier of the current host in hexadecimal. This numeric value is unique across all Sun hosts.

**SEE ALSO**

**gethostid**(2)

**NAME**

hostname – set or print name of current host system

**SYNOPSIS**

**hostname** [ *name-of-host* ]

**DESCRIPTION**

The **hostname** command prints the name of the current host, as given before the "login" prompt. The super-user can set the hostname by giving an argument; this is usually done in the startup script **/etc/rc.local**.

**FILES**

**/etc/rc.local**

**SEE ALSO**

**gethostname**(2)

## NAME

iconedit – create and edit images for SunView icons, cursors and panel items

## SYNOPSIS

**iconedit** [ *filename* ]

## OPTIONS

**iconedit** accepts the standard SunView command-line arguments; see **sunview**(1) for a list.

## AVAILABILITY

This command is available with the *SunView User's* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**iconedit** is a standard tool provided with the SunView environment. With it you can create and edit small images for use in icons, cursors, panel items, etc. **iconedit** has several subwindows:

- A large drawing area or *canvas* (on the left).
- A small proof area for previewing a life-size version of the image being edited (at the lower right).
- A control panel showing the options available and their current state (at the center right).
- An area for status messages (at the upper right).
- An area containing instructions for the use of the mouse (above the drawing canvas).

Inside the canvas, use the left button to draw and the middle button to erase. As you draw, an enlarged version of the image appears in the canvas, while a life-sized version of the image appears in the proof area. Use the right button to undo the previous operation.

While editing a cursor image, you can try the cursor out against different backgrounds and with different raster operations by moving the cursor into the proof area.

## CONTROL PANEL

The large control panel to the right of the canvas contains many items through which you can control **iconedit**. Some items are buttons which allow you to initiate commands, some are text fields which you type into, and some are choice items allowing you select from a range of options. Use the left button to select items. Most items also have a menu which you can invoke with the right button.

There are three text fields: the two at the top labeled **Dir:** and **File:**, and one to the right of the **abc** labeled **Fill:**. A triangular caret points to the current type-in position. Typing RETURN advances the caret to the next text field; you can also move the caret to a text field by selecting the field with the left button.

Each item in the control panel is described below:

**Dir** The current directory.

**File** The current filename. The default is the filename given on the command line. You can request filename completion by pressing ESC. **iconedit** searches the current directory for files whose names begin with the string you entered. If the filename search locates only one file, that file will be loaded in. In addition, typing CTRL-L, CTRL-S, CTRL-B or CTRL-Q are equivalent to pressing the **Load**, **Store**, **Browse**, or **Quit** buttons, respectively.

**Load** **(Button)** Load the canvas from the file named in the **File** field.

**Store** **(Button)** Store the current image in the file named in the **File** field.

**Browse** **(Button)** Display all the images in the current directory in a popup panel. When you select an image with the left button, it will be loaded into the canvas for editing and the browsing panel will be hidden. Pressing browse again will cause the panel to popup again (it will come up immediately if the directory and file fields have not been modified).

**Quit** **(Button)** Terminate processing. Quitting requires confirmation.

**Size** Alter the canvas size. Choices are icon size (64 x 64 pixels) or cursor size (16 x 16 pixels).

**Grid** Display a grid over the drawing canvas, or turn the grid off.

**Clear** **(Button)** Clear the canvas.

**Fill** **(Button)** Fill canvas with current rectangular fill pattern.

**Invert** **(Button)** Invert each pixel represented on the canvas.

**Paintbrush**

Select from among five painting modes. Instructions for each painting mode appear above the canvas. The painting modes are:

**dot** Paint a single dot at a time.

**line** Draw a line. To draw a line on the canvas, point to the first endpoint of the line, and press and hold the left mouse button. While holding the button down, drag the cursor to the second endpoint of the line. Release the mouse button.

**rectangle**

Draw a rectangle. To draw a rectangle on the canvas, point to the first corner of the rectangle and press and hold the left mouse button. While holding the button down, drag the cursor to the diagonally opposite corner of the rectangle. Release the mouse button.

In the control panel, the **Fill** field to the right of the rectangle indicates the current rectangle fill pattern. Any rectangles you paint on the canvas will be filled with this pattern.

**circle** Draw a circle. To draw a circle on the canvas, point to the center of the circle, and press and hold the left mouse button. While holding the button down, drag the cursor to the desired edge of the circle. Release the mouse button.

In the control panel, the **Fill** field to the right of the circle indicates the current circle fill pattern. Any circles you paint on the canvas will be filled with this pattern.

**abc** Insert text. To insert text, move the painting hand to **abc** and type the desired text. Then move the cursor to the canvas and press and hold the left mouse button. A box will appear where the text is to go. Position the box as desired and release the mouse button.

In addition, you can choose the font in which to draw the text. Point at the Fill field to the right of the **abc** and either click the left mouse button to cycle through the available fonts or press and hold the right mouse button to bring up a menu of fonts.

**Load** This is the rasterop to be used when loading a file in from disk. (See the *Pixrect Reference Manual* for details on rasterops).

**Fill** This is the rasterop to be used when filling the canvas. The source for this operation is the rectangle fill pattern, and the destination is the canvas.

**Proof** This is the rasterop to be used when rendering the proof image. The source for this operation is the proof image, and the destination is the proof background.

Proof background

The proof background can be changed to allow you to preview how the image will appear against a variety of patterns. The squares just above the proof area show the patterns available for use as the proof background pattern. To change the proof background, point at the desired pattern and click the left mouse button.

**ENVIRONMENT**

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **iconedit**. On entry to **iconedit**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default.** When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

**SEE ALSO**

**sunview**(1)

*Pixrect Reference Manual*

**NAME**

id – print the user name and ID, and group name and ID

**SYNOPSIS**

**id**

**AVAILABILITY**

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**id** displays your user and group ID, and your username. If your real and effective IDs do not match, both are printed.

**SEE ALSO**

**getuid**(2V)

## NAME

indent – indent and format a C program source file

## SYNOPSIS

**indent** *input-file* [ *output-file* ] [ [ **–bap** | **–nbap** ] [ **–bacc** | **–nbacc** ] [ **–bad** | **–nbad** ] [ **–bbb** | **–nbbb** ] [ **–bc** | **–nbc** ] [ **–bl** ] [ **–br** ] [ **–bs** | **–nbs** ] [ **–c***n* ] [ **–cd***n* ] [ **–cdb** | **–ncdb** ] [ **–ce** | **–nce** ] [ **–ci***n* ] [ **–cli***n* ] [ **–d***n* ] [ **–di***n* ] [ **–eei** | **–neei** ] [ **–fc1** | **–nfc1** ] [ **–i***n* ] [ **–ip** | **–nip** ] [ **–l***n* ] [ **–lc***n* ] [ **–lp** | **–nlp** ] [ **–pcs** | **–npcs** ] [ **–npro** ] [ **–psl** | **–npsl** ] [ **–sc** | **–nsc** ] [ **–sob** | **–nsob** ] [ **–st** ] [ **–troff** ] [ **–v** | **–nv** ]

## DESCRIPTION

**indent** is a C program formatter. It reformats the C program in the *input-file* according to the switches. The switches which can be specified are described below. They may appear before or after the file names.

Note: if you only specify an *input-file*, the formatting is done "in-place", that is, the formatted file is written back into *input-file* and a backup copy of *input-file* is written in the current directory. If *input-file* is named **/blah/blah/file**, the backup file is named **file.BAK**.

If *output-file* is specified, **indent** checks to make sure it is different from *input-file*.

## OPTIONS

The options listed below control the formatting style imposed by **indent**.

**–bap,–nbap** If **–bap** is specified, a blank line is forced after every procedure body. Default: **–nbap.**

**–bacc,–nbacc** If **–bacc** is specified, a blank line is forced around every conditional compilation block. That is. in front of every **#ifdef** and after every **#endif**. Other blanklines surrounding these will be swallowed. Default: **–nbacc.**

**–bad,–nbad** If **–bad** is specified, a blank line is forced after every block of declarations. Default: **–nbad.**

**–bbb,–nbbb** If **–bbb** is specified, a blank line is forced before every block comment. Default: **–nbbb.**

**–bc,–nbc** If **–bc** is specified, then a NEWLINE is forced after each comma in a declaration. **–nbc** turns off this option. The default is **–bc.**

**–br,–bl** Specifying **–bl** lines up compound statements like this:

```
if (...)
{
                    code
}
```

Specifying **–br** (the default) makes them look like this:

```
if (...) {
                    code
}
```

**–bs,–nbs**

Enable (disable) the forcing of a blank after **sizeof**. Some people believe that **sizeof** should appear as though it were a procedure call (**–nbs**, the default) and some people believe that since **sizeof** is an operator, it should always be treated that way and should always have a blank after it.

**–c***n* The column in which comments on code start. The default is 33.

**–cd***n* The column in which comments on declarations start. The default is for these comments to start in the same column as those on code.

**-cdb,-ncdb**

Enable (disable) the placement of comment delimiters on blank lines. With this option enabled, comments look like this:

```
/*
* this is a comment
*/
```

Rather than like this:

```
/* this is a comment */
```

This only affects block comments, not comments to the right of code. The default is **-cdb**.

**-ce,-nce**

Enables (disables) forcing **else**'s to cuddle up to the immediately preceding '}'. The default is **-ce**.

**-ci***n* Sets the continuation indent to be *n*. Continuation lines will be indented that far from the beginning of the first line of the statement. Parenthesized expressions have extra indentation added to indicate the nesting, unless **-lp** is in effect. **-ci** defaults to the same value as **-i**.

**-cli***n* Cause case labels to be indented *n* tab stops to the right of the containing **switch** statement. **-cli0.5** causes case labels to be indented half a tab stop. The default is **-cli0**.

**-d***n* Control the placement of comments which are not to the right of code. The default **-d1** means that such comments are placed one indentation level to the left of code. Specifying **-d0** lines up these comments with the code. See the section on comment indentation below.

**-di***n* Specify the indentation, in character positions, from a declaration keyword to the following identifier. The default is **-di16**.

**-eei,-neei**

If **-eei** is specified, and extra expression indent is applied on continuation lines of the expression part of **if**( ) and **while**( ). These continuation lines will be indented one extra level — twice instead of just once. This is to avoid the confusion between the continued expression and the statement that follows the **if**( ) or **while**( ). Default: **-neei.**

**-fc1,-nfc1**

Enables (disables) the formatting of comments that start in column 1. Often, comments whose leading '/' is in column 1 have been carefully hand formatted by the programmer. In such cases, **-nfc1** should be used. The default is **-fc1.**

**-i***n* The number of spaces for one indentation level. The default is 4.

**-ip,-nip**

Enables (disables) the indentation of parameter declarations from the left margin. The default is **-ip**.

**-l***n* Maximum length of an output line with a trailing comment. The default is 78.

**-lc***n* Sets the line length for block comments to *n*. It defaults to being the same as the usual line length as specified with **-l.**

**-lp,-nlp**

Lines up code surrounded by parenthesis in continuation lines. If a line has a left paren which is not closed on that line, then continuation lines will be lined up to start at the character position just after the left parenthesis. For example, here is how a piece of continued code looks with **-nlp** in effect:

```
p1 = first_procedure(second_procedure(p2, p3),
                     third_procedure(p4, p5));
```

With **-lp** in effect (the default) the code looks somewhat clearer:

```
p1 = first_procedure(second_procedure(p2, p3),
                     third_procedure(p4, p5));
```

Inserting a couple more NEWLINE characters we get:

```
p1 = first_procedure(second_procedure(p2,
                                      p3),
                     third_procedure(p4,
                                     p5));
```

**-npro** Ignore the profile files, **./.indent.pro** and **~/.indent.pro**.

**-pcs , -npcs**
If true (**-pcs**) all procedure calls will have a space inserted between the name and the '('. The default is **-npcs**.

**-psl , -npsl**
If true (**-psl**) the names of procedures being defined are placed in column 1 — their types, if any, will be left on the previous lines. The default is **-psl**.

**-sc,-nsc**
Enables (disables) the placement of asterisks ('*'s) at the left edge of all comments.

**-sob,-nsob**
If **-sob** is specified, indent will swallow optional blank lines. You can use this to get rid of blank lines after declarations. The default is **-nsob**.

**-st** **indent** takes its input from the standard input, and put its output to the standard output.

**-T** *typename*
Add *typename* to the list of type keywords. Names accumulate: **-T** can be specified more than once. You need to specify all the typenames that appear in your program that are defined by **typedefs** — nothing will be harmed if you miss a few, but the program won't be formatted as nicely as it should. This sounds like a painful thing to have to do, but it is really a symptom of a problem in C: **typedef** causes a syntactic change in the language and *indent* cannot find all **typedefs**.

**-troff** Causes **indent** to format the program for processing by **troff**. It will produce a fancy listing in much the same spirit as **vgrind**. If the output file is not specified, the default is the standard output, rather than formatting in place. The usual way to get a **troff**ed listing is with the command

**indent -troff** *program.c* **| troff -mindent**

**-v,-nv** **-v** turns on "verbose" mode, **-nv** turns it off. When in verbose mode, **indent** reports when it splits one line of input into two or more lines of output, and gives some size statistics at completion. The default is **-nv**.

## FURTHER DESCRIPTION

You may set up your own "profile" of defaults to **indent** by creating a file called **.indent.pro** in either your login directory or the current directory and including whatever switches you like. An **.indent.pro** in the current directory takes precedence over the one in your login directory. If **indent** is run and a profile file exists, then it is read to set up the program's defaults. Switches on the command line, though, always override profile switches. The switches should be separated by SPACE, TAB, or NEWLINE characters.

### Comments

Boxed **indent** assumes that any comment with a dash or star immediately after the start of comment (that is, '/*-'or'/**') is a comment surrounded by a box of stars. Each line of such a comment is left unchanged, except that its indentation may be adjusted to account for the change in indentation of the first line of the comment.

Straight text — All other comments are treated as straight text. **indent** fits as many words (separated by SPACE, TAB, or NEWLINE characters) on a line as possible. Blank lines break paragraphs.

**Comment indentation**

If a comment is on a line with code it is started in the "comment column", which is set by the **–c***n* command line parameter. Otherwise, the comment is started at *n* indentation levels less than where code is currently being placed, where *n* is specified by the **–d***n* command line parameter. If the code on a line extends past the comment column, the comment starts further to the right, and the right margin may be automatically extended in extreme cases.

**Preprocessor lines**

In general, **indent** leaves preprocessor lines alone. The only reformatting that it will do is to straighten up trailing comments. It leaves imbedded comments alone. Conditional compilation (**#ifdef...#endif**) is recognized and **indent** attempts to correctly compensate for the syntactic peculiarities introduced.

**C syntax**

**indent** understands a substantial amount about the syntax of C, but it has a "forgiving" parser. It attempts to cope with the usual sorts of incomplete and misformed syntax. In particular, the use of macros like:

```
#define forever for(;;)
```

is handled properly.

## FILES

| | |
|---|---|
| **./.indent.pro** | profile file |
| **~/.indent.pro** | profile file |
| **/usr/share/lib/tmac/tmac.indent** | troff macro package for '**indent –troff**' output. |

## SEE ALSO

ls(1V), troff(1)

## BUGS

**indent** has even more switches than ls(1V).

A common mistake that often causes grief is typing:

```
indent *.c
```

to the shell in an attempt to indent all the C programs in a directory. This is probably a bug, not a feature.

The **–bs** option splits an excessively fine hair.

## NAME

indxbib – create an inverted index to a bibliographic database

## SYNOPSIS

**indxbib** *database-file...*

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**indxbib** makes an inverted index to the named *database-file* (which must reside within the current directory), typically for use by **lookbib**(1) and **refer**(1). A *database* contains bibliographic references (or other kinds of information) separated by blank lines.

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a '**%**', followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with '**%**'.

**indxbib** is a shell script that calls two programs: **/usr/lib/refer/mkey** and **/usr/lib/refer/inv. mkey** truncates words to 6 characters, and maps upper case to lower case. It also discards words shorter than 3 characters, words among the 100 most common English words, and numbers (dates) < 1900 or > 2000. These parameters can be changed. See **refer** in *Formatting Documents* for details.

**indxbib** creates an entry file (with a **.ia** suffix), a posting file (**.ib**), and a tag file (**.ic**), in the working directory.

## FILES

**/usr/lib/refer/mkey**
**/usr/lib/refer/inv**

| | |
|---|---|
| *x*.**ia** | entry file |
| *x*.**ib** | posting file |
| *x*.**ic** | tag file |
| *x*.**ig** | reference file |

## SEE ALSO

**addbib**(1), **lookbib**(1), **refer**(1), **roffbib**(1), **sortbib**(1)

*Formatting Documents*

## BUGS

All dates should probably be indexed, since many disciplines refer to literature written in the 1800s or earlier.

**indxbib** does not recognize pathnames.

## NAME

inline – in-line procedure call expander

## SYNOPSIS

**/usr/lib/inline** [ **–w** ] [ **–v** ] [ **–o** *outputfile* ] [ **–i** *inlinefile* ] ... [ *cpu-option* ] [ *fpu-option* ] *filename*...

## DESCRIPTION

**inline** expands assembly language calls in the indicated source files into copies of the corresponding procedure bodies obtained from an *inlinefile* specified with the **–i** option. If no *inlinefile* is specified, the source files are simply concatenated and written to the standard output. If no source files are specified, the input is read from the standard input.

Inline itself is little more than a sed script. Almost all of the benefit produced is derived from subsequent peephole optimization.

## OPTIONS

**–w** Display warnings for duplicate definitions on the standard error.

**–v** Verbose. Display the names of routines that were actually in-line expanded in the sourcefile on the standard error.

**–o** *outputfile*
write output to the indicated file; standard output by default.

**–i** *inlinefile*
Read in-line code templates from *inlinefile*.

*cpu-option*
Specify templates for the machine architecture of a Sun-2 or Sun-3 system. If this option is omitted, the proper template for the host architecture is used. Can be one of:

**–mc68010** expand **.mc68010** code templates
**–mc68020** expand **.mc68020** code templates

*fpu-option*
Specify a floating-point processor option for a Sun-2 or Sun-3 system. Can be one of:

**–fsoft** expand **.fsoft** code templates (the default)
**–fswitch** expand **.fswitch** code templates
**–fsky** expand **.fsky** code templates (**–mc68010** only)
**–f68881** expand **.f68881** code templates (**–mc68020** only)
**–ffpa** expand **.ffpa** code templates (**–mc68020** only)

## USAGE

Each *inlinefile* contains one or more labeled assembly language templates of the form:

*inline-directive*
...
*instructions*
...
*.end*

where the *instructions* constitute an in-line expansion of the named routine. An *inline-directive* is a command of the form:

**.inline** *identifier*, *argsize*

This declares a block of code for the routine named by *identifier*, with *argsize* bytes of arguments. (*argsize* is optional on Sun-4 systems). Calls to the named routine are replaced by the code in the in-line template.

For Sun-2 and Sun-3 systems, the following additional forms are recognized:

| | |
|---|---|
| **.mc68010** | *identifier, argsize* |
| **.mc68020** | *identifier, argsize* |
| **.fsoft** | *identifier, argsize* |
| **.fswitch** | *identifier, argsize* |
| **.fsky** | *identifier, argsize* |
| **.f68881** | *identifier, argsize* |
| **.ffpa** | *identifier, argsize* |

These forms are similar to **.inline**, with the addition of a CPU or FPU specification. The template is only expanded if the specified target system matches the value of the target CPU or FPU type, as determined by the command-line options, or if none were given, by the type of the host system.

Multiple templates are permitted; matching templates after the first are ignored. Duplicate templates may be placed in order of decreasing performance of the corresponding hardware; thus the most efficient usable version will be selected.

**Coding Conventions for all Sun Systems**

In-line templates should be coded as expansions of C-compatible procedure calls, with the difference that the return address cannot be depended upon to be in the expected place, since no call instruction will have been executed. See FILES, below, for examples.

In-line templates must conform to standard Sun parameter passing and register usage conventions, as detailed below. They must not call routines that violate these conventions; for example, assembly language routines such as **setjmp**(3V) may cause problems.

Registers other than the ones mentioned below must not be used or set.

Branch instructions in an in-line template may only transfer to numeric labels (**1f**, **2b**, and so on) defined within the in-line template. No other control transfers are allowed.

Only opcodes and addressing modes generated by Sun compilers are guaranteed to work. Binary encodings of instructions are not supported.

**Coding Conventions for Sun-2 and Sun-3**

Arguments are passed in 32-bit aligned memory locations starting at **sp@**. Note that there is no return address on the stack, since no **jbsr** instruction will have been executed.

Results are returned in **d0** or **d0/d1**.

The following registers may be used as temporaries: registers **a0**, **a1**, **d0**, and **d1** on the MC68010 and MC68020; registers **fp0** and **fp1** on the MC68881; registers **fpa0** through **fpa3** on the Sun Floating-Point Accelerator. No other registers may be used.

The template must delete exactly *argsize* bytes from the stack. This is to enable **inline** to deal with autoincrement and autodecrement addressing modes, which in turn are used by **c2** to delimit the lifetimes of stack temporaries.

The stack must not underflow the level of the last argument.

Use **j***cc* branch mnemonics instead of **b***cc*. The **b***cc* ops are span limited and will fail if retargeted to a label whose span overflows the branch displacement field.

**Coding Conventions for Sun-4 Systems**

Arguments are passed in registers **%o0-%o5**, followed by memory locations starting at **[%sp+0x5c]**. **%sp** is guaranteed to be 64-bit aligned. The contents of **%o7** are undefined, since no call instruction will have been executed.

Results are returned in **%o0** or **%f0/%f1**.

Registers **%o0-%o5** and **%f0-%f31** may be used as temporaries.

Integral and single-precision floating-point arguments are 32-bit aligned.

Double-precision floating-point arguments are guaranteed to be 64-bit aligned if their offsets are multiples of 8.

Each control-transfer instruction (branches and calls) must be immediately followed by a nop.

Call instructions must include an extra (final) argument which indicates the number of registers used to pass parameters to the called routine.

Note that for Sun-4 systems, the instruction following an expanded 'call' is inserted by **inline** *before* the expanded code to preserve the semantics of the call's delay slot.

**FILES**

| | |
|---|---|
| **/usr/lib/inline** | in-line procedure call expander |
| **/usr/lib/fsoft/libm.il** | in-line templates for software floating point (Sun-2 and Sun-3 only) |
| **/usr/lib/fswitch/libm.il** | in-line templates for switched floating point (Sun-2 and Sun-3 only) |
| **/usr/lib/fsky/libm.il** | in-line templates for Sky FFP (Sun-2 only) |
| **/usr/lib/f68881/libm.il** | in-line templates for Motorola 68881 (Sun-3 only) |
| **/usr/lib/ffpa/libm.il** | in-line templates for Sun FPA (Sun-3 only) |

**WARNING**

**inline** does not check for violations of the coding conventions described above.

## NAME

input_from_defaults, defaults_from_input – update the current state of the mouse and keyboard from the defaults database, and vice versa

## SYNOPSIS

**input_from_defaults**
**defaults_from_input**

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**input_from_defaults** updates various parameters controlling mouse- and keyboard-processing on the machine on which it is run. It should be used on systems which are running the SunView window system. The parameters control the distribution of function keys on the keyboard, the assignment of buttons on the mouse, the scaling of mouse-to-cursor motion, and the effect of two filters on mouse-motion originally provided to compensate for defective mice. The new values are taken from the defaults database, starting with the file **.defaults** in the user's home directory. When **Private_only** is **False** in the **$HOME/.defaults** file, the **/usr/lib/defaults/*.d** file is consulted. See **defaultsedit**(1) for more information about the defaults database.

**defaults_from_input** is the inverse operation to **input_from_defaults**. It updates the user's private defaults database (used by **defaultsedit**(1)) to reflect the current state of kernel input parameters listed above.

## FILES

**$HOME/.defaults**
**/usr/lib/defaults/*.d**
**$HOME/.login**

## SEE ALSO

**csh**(1), **defaultsedit**(1)

*SunView User's Guide*

## NOTES

The parameter settings set by **input_from_defaults** are lost when the system goes down. To set them automatically on login, **csh**(1) users can include the following in their **.login** files:

```
if ('tty'==/dev/console) then
        echo 'Setting input defaults'
        input_from_defaults
endif
```

## BUGS

**input_from_defaults** should be targetable to any user's **.defaults** file.

## NAME

install – install files

## SYNOPSIS

**install** [ **–cs** ] [ **–g** *group* ] [ **–m** *mode* ] [ **–o** *owner* ] *file1 file2*

**install** [ **–cs** ] [ **–g** *group* ] [ **–m** *mode* ] [ **–o** *owner* ] *file ... directory*

**install** **–d** [ **–g** *group* ] [ **–m** *mode* ] [ **–o** *owner* ] *directory*

## AVAILABILITY

This command is available with the *Install* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

Install is used within makefiles to copy new versions of files into a destination directory and to create the destination directory itself.

The first two forms are similar to the **cp**(1) command with the addition that executable files can be stripped during the copy and the owner, group, and mode of the installed file(s) can be given.

The third form can be used to create a destination directory with the required owner, group and permissions.

Note: **install** uses no special privileges to copy files from one place to another. The implications of this are:

- You must have permission to read the files to be installed.
- You must have permission to copy into the destination file or directory.
- You must have permission to change the modes on the final copy of the file if you want to use the **–m** option to change modes.
- You must be superuser if you want to specify the ownership of the installed file with **–o**. If you are not the super-user, or if **–o** is not in effect, the installed file will be owned by you, regardless of who owns the original.

## OPTIONS

| | |
|---|---|
| **–g** *group* | Set the group ownership of the installed file or directory. (staff by default) |
| **–m** *mode* | Set the mode for the installed file or directory. (0755 by default) |
| **–o** *owner* | If run as root, set the ownership of the installed file to the user-ID of *owner*. |
| **–c** | Copy files. In fact **install** *always* copies files, but the **–c** option is retained for backwards compatibility with old shell scripts that might otherwise break. |
| **–s** | Strip executable files after they are copied. |
| **–d** | Create a directory. Missing parent directories are created as required as in **mkdir –p**. If the directory already exists, the owner, group and mode will be set to the values given on the command line. |

## SEE ALSO

**chgrp**(1), **chmod**(1V), **cp**(1), **mkdir**(1), **strip**(1), **chown**(8)

**NAME**

ipcrm – remove a message queue, semaphore set, or shared memory ID

**SYNOPSIS**

**ipcrm** [ *primitives* ]

**DESCRIPTION**

**ipcrm** removes one or several messages, semaphores, or shared memory identifiers, as specified by the following *primitives*:

**–q** *msqid*
removes the message queue identifier *msqid* from the system and destroys the message queue and data structures associated with it.

**–m** *shmid*
removes the shared memory identifier *shmid* from the system. The shared memory segment and data structures associated with it are destroyed after the last detach.

**–s** *semid*
removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structures associated with it.

**–Q** *msgkey*
removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structures associated with it.

**–M** *shmkey*
removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structures associated with it are destroyed after the last detach.

**–S** *semkey*
removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structures associated with it.

The identifiers and keys may be found by using **ipcs**(1).

The details of removing identifiers are described in **msgctl**(2), **shmctl**(2), and **semctl**(2) in the sections detailing the **IPC_RMID** command.

**SEE ALSO**

**ipcs**(1), **msgctl**(2), **msgget**(2), **semctl**(2), **semget**(2), **semop**(2), **shmctl**(2), **shmget**(2), **shmop**(2)

**NAME**

ipcs – report interprocess communication facilities status

**SYNOPSIS**

**ipcs** [ *primitives* ]

**DESCRIPTION**

**ipcs** prints information about active interprocess communication facilities as specified by the *primitives* shown below. If no *primitives* are given, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system.

**Command – Line Primitives**

If any of the *primitives* **–m**, **–q**, or **–s** are specified, information about only indicated facilities is printed. If none of these are specified, information about all three is printed.

**–q** Print information about active message queues.

**–m** Print information about active shared memory segments.

**–s** Print information about active semaphores.

**–b** Print the currently allowed size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for the meaning of columns in a listing.

**–c** Print creator's login name and group name. See below.

**–o** Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

**–p** Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments)See below.

**–t** Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last **msgsnd** and last **msgrcv** (see **msgop**(2)) on message queues, last **shmat** and last **shmdt** (see **shmop**(2)) on shared memory, last **semop**(2) on semaphores.) See below.

**–a** Use all display *primitive*s. (This is a shorthand notation for **–b**, **–c**, **–o**, **–p**, and **–t**.)

**–C** *corefile* Use the file *corefile* in place of **/dev/mem**.

**–N** *namelist* The argument will be taken as the name of an alternate *filenamelist* (**/vmunix** is the default).

The column headings and the meaning of the columns in an **ipcs** listing are given below; the letters in parentheses indicate the *primitives* that cause the corresponding heading to appear; **all** means that the heading always appears. Note: these *primitives* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

**T** (all) Type of the facility:
- **q** message queue
- **m** shared memory segment
- **s** semaphore

**ID** (all) The identifier for the facility entry.

**KEY** (all) The key used as an argument to **msgget**(2), **semget**(2), or **shmget**(2) to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

**MODE** (all) The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:

The first two characters are:

**R** If a process is waiting on a **msgrcv**.
**S** If a process is waiting on a **msgsnd**.
**D** If the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it.
**C** If the associated shared memory segment is to be cleared when the first attach is executed.
– If the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

**r** If read permission is granted.
**w** If write permission is granted.
**a** If alter permission is granted.
— If the indicated permission is *not* granted.

**OWNER** (all) The login name of the owner of the facility entry.

**GROUP** (all) The group name of the group of the owner of the facility entry.

**CREATOR** (a,c) The login name of the creator of the facility entry.

**CGROUP** (a,c) The group name of the group of the creator of the facility entry.

**CBYTES** (a,o) The number of bytes in messages currently outstanding on the associated message queue.

**QNUM** (a,o) The number of messages currently outstanding on the associated message queue.

**QBYTES** (a,b) The maximum number of bytes allowed in messages outstanding on the associated message queue.

**LSPID** (a,p) The process ID of the last process to send a message to the associated queue.

**LRPID** (a,p) The process ID of the last process to receive a message from the associated queue.

**STIME** (a,t) The time the last message was sent to the associated queue.

**RTIME** (a,t) The time the last message was received from the associated queue.

**CTIME** (a,t) The time when the associated entry was created or changed.

**NATTCH** (a,o) The number of processes attached to the associated shared memory segment.

**SEGSZ** (a,b) The size of the associated shared memory segment.

**CPID** (a,p) The process ID of the creator of the shared memory entry.

**LPID** (a,p) The process ID of the last process to attach or detach the shared memory segment.

**ATIME** (a,t) The time the last attach was completed to the associated shared memory segment.

**DTIME** (a,t) The time the last detach was completed on the associated shared memory segment.

**NSEMS** (a,b) The number of semaphores in the set associated with the semaphore entry.

**OTIME** (a,t) The time the last semaphore operation was completed on the set associated with the semaphore entry.

**FILES**

| | |
|---|---|
| **/vmunix** | system namelist |
| **/dev/mem** | memory |
| **/etc/passwd** | user names |
| **/etc/group** | group names |

**SEE ALSO**

**ipcrm**(1), **msgop**(2), **semctl**(2), **semget**(2), **semop**(2), **shmctl**(2), **shmget**(2), **shmop**(2)

**BUGS**

Things can change while **ipcs** is running; the picture it gives is only a close approximation to reality.

**NAME**

join – relational database operator

**SYNOPSIS**

**join** [ **–a***n* ] [ **–e** *string* ] [ **–j** [**1**|**2**] *m* ] [ **–o** *list* ] [ **–t***c* ] *filename1 filename2*

**DESCRIPTION**

**join** forms, on the standard output, a join of the two relations specified by the lines of *filename1* and *filename2*. If *filename1* is '–', the standard input is used.

*filename1* and *filename2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined — normally the first in each line.

There is one line in the output for each pair of lines in *filename1* and *filename2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *filename1*, then the rest of the line from *filename2*.

The default input field separators are SPACE, TAB, and NEWLINE characters. If the default input field separators are used, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

**OPTIONS**

**–a***n* The parameter *n* can be one of the values:

1 Produce a line for each unpairable line in *filename1*.

2 Produce a line for each unpairable line in *filename2*.

3 Produce a line for each unpairable line in both *filename1* and *filename2*.

The normal output is also produced.

**–e** *string*

Replace empty output fields by *string*.

**–j**[**1**|**2**]*m*

The **j** may be immediately followed by *n*, which is either a **1** or a **2**. If *n* is missing, the join is on the *m*'th field of both files. If *n* is present, the join is on the *m*'th field of file *n*, and the first field of the other. Note: **join** counts fields from 1 instead of 0 as **sort**(1V) does.

**–o** *list* Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested. Note: **join** counts fields from 1 instead of 0 like **sort** does.

**–t***c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

**EXAMPLE**

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

**join –j1 4 –j2 3 –o 1.1 2.1 1.6 –t: /etc/passwd /etc/group**

**SEE ALSO**

**awk**(1), **comm**(1), **look**(1), **sort**(1V), **uniq**(1)

**BUGS**

With default field separation, the collating sequence is that of **sort –b**; with **–t**, the sequence is that of a plain sort.

The conventions of **join**, **sort**, **comm**, **uniq**, **look**, and **awk** are wildly incongruous.

Filenames that are numeric may cause conflict when the **–o** option is used right before listing filenames.

**NAME**

keylogin – decrypt and store secret key

**SYNOPSIS**

**keylogin**

**DESCRIPTION**

**keylogin** prompts the user for their login password, and uses it to decrypt the user's secret key stored in the **publickey**(5) database. Once decrypted, the user's key is stored by the local key server process **keyserv**(8C) to be used by any secure network services, such as NFS.

Normally, **login**(1) does this work when the user logs onto the system, but running **keylogin** may be necessary if the user did not type a password to **login**(1).

**SEE ALSO**

**chkey**(1), **login**(1), **publickey**(5), **keyserv**(8C), **newkey**(8)

**NAME**

keylogout – delete stored secret key

**SYNOPSIS**

**keylogout** [ **–f** ]

**DESCRIPTION**

**keylogout** deletes the key stored by the key server process **keyserv**(8C) to be used by any secure network services, such as NFS. Further access to the key is revoked, however current session keys may remain valid till they expire, or are refreshed. This option will cause any background jobs that need secure RPC services to fail, and any scheduled at jobs that need the key to fail. Also since only one copy is kept on a machine of the key, it is a bad idea to place this in your **.logout** file since it will affect other sessions on the same machine.

**OPTIONS**

**–f** Forget the rootkey. This will break secure NFS if it is done on a server.

**SEE ALSO**

**chkey**(1), **login**(1), **keylogin**(1), **publickey**(5), **keyserv**(8C), **newkey**(8)

**NAME**

kill – send a signal to a process, or terminate a process

**SYNOPSIS**

**kill** [ *–signal* ] *pid* ...
**kill –l**

**DESCRIPTION**

**kill** sends the **TERM** (terminate, 15) signal to the processes with the specified *pid*s. If a signal name or number preceded by '–' is given as first argument, that signal is sent instead of terminate. The signal names are listed by using the **–l** option, and are as given in **<signal.h>**, stripped of the common **SIG** prefix.

The terminate signal will kill processes that do not catch the signal, so '**kill –9 ...**' is a sure kill, as the **KILL (9)** signal cannot be caught. By convention, if process number 0 is specified, all members in the process group (that is, processes resulting from the current login) are signaled (but beware: this works only if you use **sh**(1); not if you use **csh**(1).) Negative process numbers also have special meanings; see **kill**(2V) for details. The killed processes must belong to the current user unless he is the super-user.

To shut the system down and bring it up single user the super-user may send the initialization process a **TERM** (terminate) signal by '**kill 1**'; see **init**(8). To force **init** to close and open terminals according to what is currently in **/etc/ttytab** use '**kill –HUP 1**' (sending a hangup signal to process 1).

The shell reports the process number of an asynchronous process started with '**&**' (run in the background). Process numbers can also be found by using **ps**(1).

**kill** is built in to **csh**(1); it allows job specifiers, such as '**kill % ...**', in place of **kill** arguments. See **csh**(1) for details.

**OPTIONS**

**–l** Display a list of signal names.

**FILES**

**/etc/ttytab**

**SEE ALSO**

**csh**(1), **ps**(1), **kill**(2V), **sigvec(2)**, **init**(8)

**BUGS**

A replacement for '**kill 0**' for **csh**(1) users should be provided.

**NAME**

last – indicate last logins by user or terminal

**SYNOPSIS**

**last** [ *–number* ][ **–f** *filename* ] [ *name...* ] [ *tty...* ]

**DESCRIPTION**

**last** looks back in the **wtmp** file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example **last 0** is the same as **last***tty0*. If multiple arguments are given, the information which applies to any of the arguments is printed. For example **last root console** would list all of "root's" sessions as well as all sessions on the console terminal. **last** displays the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, **last** so indicates.

The pseudo-user **reboot** logs in at reboots of the system, thus

**last reboot**

will give an indication of mean time between reboot.

**last** with no arguments displays a record of all logins and logouts, in reverse order.

If **last** is interrupted, it indicates how far the search has progressed in **wtmp.** If interrupted with a quit signal (generated by a CTRL-\) **last** indicates how far the search has progressed so far, and the search continues.

**OPTIONS**

*–number*

limit the number of entries displayed to that specified by *number*.

**–f** *filename*

Use *filename* as the name of the accounting file instead of **/var/adm/wtmp.**

**FILES**

**/var/adm/wtmp** login data base

**SEE ALSO**

**lastcomm**(1), **utmp**(5V), **ac**(8)

**NAME**

lastcomm – show the last commands executed, in reverse order

**SYNOPSIS**

**lastcomm** [ *command-name* ] ... [ *user-name* ] ... [ *terminal-name* ] ...

**DESCRIPTION**

**lastcomm** gives information on previously executed commands. **lastcomm** with no arguments displays information about all the commands recorded during the current accounting file's lifetime. If called with arguments, **lastcomm** displays only accounting entries with a matching *command-name*, *user-name*, or *terminal-name*.

**EXAMPLES**

The command

**example% lastcomm a.out root ttyd0**

would produce a listing of all the executions of commands named **a.out**, by user **root** while using the terminal **ttyd0**.

**example% lastcomm root**

would produce a listing of all the commands executed by user **root**.

For each process entry, **lastcomm** displays the following information:

- The command name under which the process was called.
- One or more flags indicating special information about the process. The flags have the following meanings:
  - **F** The process performed a **fork** but not an **exec**.
  - **S** The process ran as a set-user-id program.
  - **D** The process dumped memory.
  - **X** The process was killed by some signal.
- The name of the user who ran the process.
- The terminal from which the user was logged in at the time (if applicable).
- The amount of CPU time used by the process (in seconds).
- The date and time the process starts.

**FILES**

**/var/adm/pacct** accounting file

**SEE ALSO**

**last**(1), **sigvec**(2), **acct**(5), **core**(5)

## NAME

ld, ld.so – link editor, dynamic link editor

## SYNOPSIS

**ld** [ **–align** *datum* ] [ **–assert** *assertion-keyword* ] [ **–A** *name* ] [ **–B***binding-keyword* ] [ **–d** ] [ **–dc** ] [ **–dp** ] [ **–D** *hex* ] [ **–e** *entry* ] [ **–lx** ] [ **–L***dir* ] [ **–M** ] [ **–n** ] [ **–N** ] [ **–o** *name* ] [ **–p** ] [ **–r** ] [ **–s** ] [ **–S** ] [ **–t** ] [ **–T** [ **text** ] *hex* ] [ **–Tdata** *hex* ] [ **–u** *name* ] [ **–x** ] [ **–X** ] [ **–y***sym* ] [ **–z** ] *filename* ...

## DESCRIPTION

**ld** combines object programs to create an *executable* file or another object program suitable for further **ld** processing (with the **–r** option). The object modules on which **ld** operates are specified on the command line, and can be:

- simple object files, which typically end in the **.o** suffix, and are referred to as "dot-oh" files
- **ar**(1V) library archives (**.a**), or "libraries"
- dynamically-bound, sharable object files (**.so**), are also referred to as "shared libraries," which are created from previous **ld** executions.

Unless an output file is specified, **ld** produces a file named **a.out**. This file contains the object files given as input, appropriately combined to form an executable file.

## OPTIONS

When linking debugging or profiling objects, include the **–g** or **–pg** option (see **cc**(1V)), as appropriate, in the **ld** command.

Options should appear before filenames, except for abbreviated library names specified with **–l** options, and some binding control options specified by **–B** (which can appear anywhere in the line).

**–align** *datum*

Force the global uninitialized data symbol *datum* (usually a FORTRAN common block) to be page-aligned. Increase its size to a whole number of pages, and place its first byte at the start of a page.

**–assert** *assertion-keyword*

Check an assertion about the link editing being performed. The assertion desired is specified by the *assertion-keyword* string. **ld** is silent if the assertion holds, else it yields a diagnostic and aborts. Valid *assertion-keyword*'s and their interpretations are:

| | |
|---|---|
| **definitions** | If the resulting program were run now, there would be no run-time undefined symbol diagnostics. This assertion is set by default. |
| **nodefinitions** | Permit the successful linking of a program with unresolved references. |
| **nosymbolic** | There are no symbolic relocation items remaining to be resolved. |
| **pure-text** | The resulting load has no relocation items remaining in its text. |

**–A** *name*

Incremental loading: linking is to be done in a manner so that the resulting object may be read into an already executing program. *name* is the name of a file whose symbol table is taken as a basis on which to define additional symbols. Only newly linked material is entered into the text and data portions of **a.out**, but the new symbol table will reflect all symbols defined before and after the incremental load. This argument must appear before any other object file in the argument list. One or both of the **–T** options may be used as well, and will be taken to mean that the newly linked segment will commence at the corresponding addresses (which must be a multiple of the page size). The default value is the old value of **_end**.

**–B***binding-keyword*

Specify allowed binding times for the items which follow. Allowed values of *binding-keyword* are:

**dynamic** Allow dynamic binding: do not resolve symbolic references, allow creation of run-time symbol and relocation environment. **-Bdynamic** is the default. When **–Bdynamic** is in effect, all sharable objects encountered until a succeeding **–Bstatic** may be added dynamically to the object being linked. Non-sharable objects are bound statically.

**nosymbolic** Do not perform symbolic relocation, even if other options imply it.

**static** Bind statically. Opposite of **–Bdynamic**. Implied when either **–n** or **–N** is specified. Influences handling of all objects following its specification on a command line until the next **–Bdynamic**.

**symbolic** Force symbolic relocation. Normally implied if an entry point has been specified with **–e**, or if dynamic loading is in effect.

**–d** Force common storage for uninitialized variables and other common symbols to be allocated in the current **ld** run, even when the **–r** flag is present (which would otherwise postpone this binding until the final linking phase).

**–dc** Do **–d**, but also copy initialized data referenced by this program from shared objects.

**–dp** Force an alias definition of undefined procedure entry points. Used with dynamic binding to improve sharing and the locality of run-time relocations.

**–D** *hex* Pad the data segment with zero-valued bytes to make it *hex* bytes long.

**–e** *entry*

Define the entry point: the *entry* argument is made the name of the entry point of the loaded program. Implies **–Bsymbolic**.

**–l***x*[*.v*] This option is an abbreviation for the library name **lib***x***.a**, where *x* is a string. **ld** searches for libraries first in any directories specified with **–L** options, then in the standard directories **/lib**, **/usr/lib**, and **/usr/local/lib**. A library is searched when its name is encountered, so the placement of a **–l** is significant. If a dynamically loadable object is found, and **–Bdynamic** is in effect at that point on the command line, then **ld** prepares to access the object for relocation at run-time. In such a case, the optional *.v* suffix can be used to indicate a specific library version.

**–L***dir* Add *dir* to the list of directories in which to search for libraries. Directories specified with **–L** are searched before the standard directories, **/lib**, **/usr/lib**, and **/usr/local/lib**. When building a program in which one or more objects are loaded when **-Bdynamic** is in effect, those directories specified by –L options will be "remembered" for use at execution time. This permits the construction of software that uses shared objects as libraries not residing in the standard locations and avoids requiring the specification of the **LD_LIBRARY_PATH** environment variable in order to execute such software. Note that such directories are retained in *exactly* the form specified in the option, which means that relative directory specifications (i.e., not beginning with "/") will be evaluated relative to the current directory when the program is *run*, not just during the operation of **ld**.

**–M** Produce a primitive load map, listing the names of the files which will be loaded.

**–n** Arrange (by giving the output file a 0410 "magic number") that when the output file is executed, the text portion will be read-only with the data areas placed at the beginning of the next address boundary following the end of the text. Implies **–Bstatic**.

**–N** Do not make the text portion read-only. (Use "magic number" 0407.) Implies **–Bstatic**.

**–o** *name*

*name* is made the name of the **ld** output file, instead of **a.out**.

**-p** Arrange for the data segment to begin on a page boundary, even if the text is not shared (with the **-N** option).

**-r** Generate relocation bits in the output file so that it can be the subject of another **ld** run. This flag also prevents final definitions from being given to common symbols, and suppresses the "undefined symbol" diagnostics.

**-s** Strip the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by **strip**(1).

**-S** Strip the output by removing all symbols except locals and globals.

**-t** Trace: display the name of each file as it is processed.

**-T** [ **text** ] *hex*
Start the text segment at location *hex*. Specifying **-T** is the same as using the **-Ttext** option.

**-Tdata** *hex*
Start the data segment at location *hex*. This option is only of use to programmers wishing to write code for PROMs, since the resulting code cannot be executed by the system.

**-u** *name*
Enter *name* as an undefined symbol. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

**-x** Preserve only global (non-**.globl**) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

**-X** Record local symbols, except for those whose names begin with **L**. This option is used by **cc** to discard internally generated labels while retaining symbols local to routines.

**-y***sym* Display each file in which *sym* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. It is usually necessary to begin *sym* with an '_', as external C, FORTRAN and Pascal variables begin with underscores.

**-z** Arrange for the process demand paged from the resulting executable file (0413 "magic number"). This is the default. Results in a (32-byte) header on the output file followed by text and data segments, each of which has a multiple of page-size bytes (being padded out with null characters in the file if necessary). With this format the first few BSS segment symbols may actually end up in the data segment; this is to avoid wasting the space resulting from rounding the data segment size. Implies **-Bdynamic**.

## USAGE

### Command Line Processing

In general, options should appear ahead of the list of files to process. Unless otherwise specified, the effect of an option covers all of **ld** operations, independent of that option's placement on the command line. Exceptions to this rule include some of the binding control options specified by '**-B**' and the abbreviated library-names specified by '**-l**'. These may appear anywhere, and their influence is dependent upon their location. Some options may be obtained from environment variables, such options are interpreted before any on the command line (see ENVIRONMENT, below).

### Object File Processing

The files specified on the command line are processed in the order listed. Information is extracted from each file, and concatenated to form the output. The specific processing performed on a given file depends upon whether it is a simple object file, a library archive, or a shared library.

Simple object (**.o**) files are concatenated to the output as they are encountered.

Library archive (**.a**) files are searched exactly once each, as each is encountered; only those archive entries matching an unresolved external reference are extracted and concatenated to the output. If a member of an archive references a symbol defined by another member of that same archive, the member making the reference must appear before the member containing the definition.

On Sun386i, a library contains a dictionary of symbols, On other Sun systems, processing library archives through **ranlib**(1) provides this dictionary. In addition, you can use **lorder**(1), in combination with **tsort**(1) to place library members in calling order (see **lorder**(1) for details), or both (for fastest symbol lookup). The first member of an archived processed by **ranlib** has the reserved name of **__.SYMDEF**, which **ld** takes to be the dictionary of all symbols defined by members of the archive.

Sharable objects (**.so**) are scanned for symbol definitions and references, but are not normally included in the output from **ld**, except in cases where a shared library exports initialized data structures and the **–dc** option is in effect. However, the occurrence of each sharable object file in the **ld** command line is noted in the resulting executable file; this notation is utilized by an execution-time variant of **ld**, **ld.so**, for *deferred* and *dynamic* loading and binding during execution. See **Execution-Time Loading**, below, for details.

The **–l** option specifies a short name for an object file or archive used as a library. The full name of the object file is derived by adding the prefix **lib** and a suffix of either **.a** or **.so**[.*v*] to indicate an **ar**(1V) archive or a shared library, respectively. The specific suffix used is determined through rules discussed in **Binding and Relocation Semantics**, below.

**ld** searches for the desired object file through a list of directories specified by **–L** options, the environment variable **LD_LIBRARY_PATH**, and finally, the built-in list of standard library directories: **/lib**, **/usr/lib**, and **/usr/local/lib**.

**Binding and Relocation Semantics**

The manner in which **ld** processes a given object file is dependent in part upon the "binding mode" in which it is operating at the time the file is encountered. This binding mode is specified by the **–B** flag, which takes the keyword arguments:

**dynamic** Allow dynamic binding, do not resolve symbolic references, and allow creation of execution-time symbol and relocation information. This is the default setting.

**static** Force static binding, implied by options that generate non-sharable executable formats.

**–Bdynamic** and **–Bstatic** may be specified several times, and may be used to toggle each other on and off. Like **–l**, the influence of each depends upon its location within the command line. When **–Bdynamic** is in effect, **–l** searches may be satisfied by the first occurrence of either form of library (**.so** or **.a**), but if both are encountered, the **.so** form is preferred. When **–Bstatic** is in effect, **ld** refuses to use any **.so** libraries it encounters; it continue searching for the **.a** form. Furthermore, an explicit request to load a **.so** file is treated as an error.

After **ld** has processed all input files and command line options, the form of the output it produces is based on the information provided in both. **ld** first tries to reduce all symbolic references to relative numerical offsets within the executable it is building. To perform this "symbolic reduction," **ld** must be able to determine that:

- all information relating to the program has been provided, in particular, no **.so** is to be added at execution time; and/or
- the program has an entry point, and symbolic reduction can be performed for all symbols having definitions existing in the material provided.

It should be noted that uninitialized "common" areas (for example, uninitialized C globals) are allocated by the link editor *after* it has collected all references. In particular, this allocation can not occur in a program that still requires the addition of information contained in a **.so** file, as the missing information may affect the allocation process. Initialized "commons" however, are allocated within the executable in which their definition appears.

After **ld** has performed all the symbolic reductions it can, it attempts to transform all relative references to absolute addresses. **ld** is able to perform this "relative reduction" only if it has been provided *some* absolute address, either implicitly through the specification of an entry point, or explicitly through **ld** command-line options. If, after performing all the reductions it can, there are no further relocations or definitions to perform, then **ld** has produced a completely linked executable.

**Execution-Time Loading**

In the event that one or more reductions can not be completed, the executable will require further link editing at execution time in order to be usable. Such executables contain an data structure identified with the symbol **__DYNAMIC**. An incompletely linked "main" program should be linked with a "bootstrap" routine that invokes **ld.so**, which uses the information contained in the main program's **__DYNAMIC** to assemble the rest of the executables constituting the entire program. A standard Sun compilation driver (such as **cc**(1V)) automatically includes such a module in each "main" executable.

When **ld.so** is given control on program startup, it finds all **.so** files specified when the program was constructed (and all **.so**'s on which they depend), and loads them into the address space. The algorithm by which such files are found mimics that used when **ld** is run, and like **ld**, can be influenced by the setting of **LD_LIBRARY_PATH** and any **–L** options specified to **ld** when the program was built. **ld.so** then completes all remaining relocations, with the exception of procedure call relocations; failure to resolve a given non-procedural relocation results in termination of the program with an appropriate diagnostic.

Procedure relocations are resolved when the referencing instruction is first executed. It should be noted that it is possible for "undefined symbol" diagnostics to be produced during program execution if a given target is not defined when referenced.

Although it is possible for binding errors to occur at execution-time, such an occurrence generally indicates something wrong in the maintenance of shared objects. **ld**'s **–assert nodefinitions** function (on by default) checks at **ld**-time whether or not an execution-time binding error would occur.

**Version Handling for Shared Libraries**

To allow the independent evolution of **.so**'s used as libraries and the programs which use them, **ld**'s handling of **.so** files found through **–l** options involves the retention and management of version control information. The **.so** files used as such "shared libraries" are post-fixed with a Dewey-decimal format string describing the version of the "library" contained in the file.

The first decimal component is called the library's "major version" number, and the second component its "minor version" number. When **ld** records a **.so** used as a library, it also records these two numbers in the database used by **ld.so** at execution time. In turn, **ld.so** uses these numbers to decide which of multiple versions of a given library is "best" or whether *any* of the available versions are acceptable. The rules are:

- Major Versions Identical: the major version used at execution time must exactly match the version found at **ld**-time. Failure to find an instance of the library with a matching major version causes a diagnostic to be issued and the program's execution to be terminated.
- Highest Minor Version: in the presence of multiple instances of libraries that match the desired major version, **ld.so** uses the highest minor version it finds. However, if the highest minor version found at execution time is less than the version found at **ld**-time, a warning diagnostic is issued; program execution continues.

The semantics of version numbers are such that major version numbers should be changed whenever interfaces are changed. Minor versions should be changed to reflect compatible updates to libraries, and programs will silently favor the highest compatible version they can obtain.

**Special Symbols**

A number of symbols have special meanings to **ld** and programs should not define these symbols. The symbols described below are those actually seen by **ld**. Note: C and several other languages prepend symbols they use with '_'.

**_etext** The first location after the text of the program.

**_edata** The first location after initialized data.

**_end** The first location after all data.

**__DYNAMIC**

Identifies an **ld**-produced data structure. It is defined with a non-zero value in executables which require execution-time link editing. By convention, if defined, it is the first symbol in the symbol table associated with an **a.out** file.

**__GLOBAL_OFFSET_TABLE_**

A position-independent reference to an **ld**-constructed table of addresses. This table is constructed from "position-independent" data references occurring in objects that have been assembled with the assembler's **–k** flag (invoked on behalf of C compilations performed with the **–pic** flag). A related table (for which no symbol is currently defined) contains a series of transfer instructions and is created from "position-independent" procedure calls or, if **–dp** is specified to **ld**, a list of undefined symbols.

Symbols in object files beginning with the letter L are taken to be local symbols and unless otherwise specified are purged from **ld** output files.

## ENVIRONMENT

**LD_LIBRARY_PATH**

A colon-separated list of directories in which to search for libraries specified with the **–l** option. Similar to the **PATH** environment variable. **LD_LIBRARY_PATH** also affects library searching during execution-time loading, causing the search to use first those directories found in the environment variable, then any directories specified by **–L** options, and finally the standard directories **/usr/lib** and **/usr/local/lib**. **NOTE:** when running a set-user- or set-group-ID program, **ld.so** will only search for libraries in directories it "trusts", which are **/usr/lib**, **/usr/5lib**, **/usr/local/lib**, and any directories specified within the executable as a result of **–L** options given when the executable was constructed.

**LD_OPTIONS**

A default set of options to **ld**. **LD_OPTIONS** is interpreted by **ld** just as though its value had been placed on the command line, immediately following the name used to invoke **ld**, as in:

**example% ld $LD_OPTIONS** ... *other-arguments* ...

Note: Environment variable-names beginning with the characters '**LD_**' are reserved for possible future enhancements to **ld**.

## FILES

| | |
|---|---|
| **/usr/lib/lib*.a** | libraries |
| **lib*.so.v** | shared libraries |
| **lib*.sa.v** | exported, initialized shared library data |
| **/usr/lib/ld.so** | execution-time **ld** |
| **/usr/lib/*crt*.o** | default program bootstraps |
| **a.out** | output file |
| **/usr/local/lib** | |

## SEE ALSO

**ar**(1V), **as**(1), **cc**(1V), **lorder**(1), **ranlib**(1), **strip**(1), **tsort**(1), **ldconfig**(8)

## BUGS

Options are being overloaded and are an inappropriate vehicle for describing to **ld** the wide variety of things it can do. There needs to be a link-editing language which can be used in the more complex situations.

The **–r** option does not properly handle programs assembled with the **–k** (position-independent) flag, invoked from **cc** with **–pic** or **–PIC**.

**NAME**

ldd – list dynamic dependencies

**SYNOPSIS**

**ldd** *filename* ...

**DESCRIPTION**

For each *filename* , **ldd** lists the dynamically loaded objects on which that *filename* depends, if any. If the dynamic dependency is a "library" (a so-called "shared library"), then both the canonical form of the library name and version and the actual pathname used to access the library are listed. For example, if a given *filename* uses a shared C library version 4, which has the name **/usr/lib/libc.so.4.9** (where 9 is the most recent revision to interface version number 4) then **ldd** *filename* will report:

**filename:**

**-lc.4 => /usr/lib/libc.so.4.9**

For each *filename* which is not an executable program, or else does not require any dynamic objects for its execution, **ldd** will issue an appropriate diagnostic.

It should be noted that although all dynamically linked programs depend on the file **/usr/lib/ld.so** , **ldd** will never report this dependency.

**SEE ALSO**

**ld**(1)

**NAME**

leave – remind you when you have to leave

**SYNOPSIS**

**leave** [[ + ] *hhmm* ]

**DESCRIPTION**

**leave** sets an alarm to a time you specify and will tell you when the time is up. **leave** waits until the specified time, then reminds you that you have to leave. You are reminded 5 minutes and 1 minute before the actual time, and at the time. After the specified time, it reminds you every minute thereafter 10 more times. **leave** disappears after you log off.

You can specify the time in on of two ways, namely as an absolute time of day in the form *hhmm* where *hh* is a time in hours (on a 12 or 24 hour clock), or you can place a + sign in front of the time, in which case the time is relative to the current time, that is, the specified number of hours and minutes from now. All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no argument is given, **leave** prompts with **'When do you have to leave?'**. **leave** exits if you just type a NEWLINE, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a **.login** or **.profile**.

**leave** ignores interrupts, quits, and terminates. To get rid of it you should either log off or use **kill –9** and its process ID.

**EXAMPLES**

The first example sets the alarm to an absolute time of day:

**example% leave 1535**
**Alarm set for Wed Mar 7 15:35:07 1984**

*work work work work*

**example% Time to leave!**

The second example sets the alarm for 10 minutes in the future:

**example% leave +10 Alarm set for Wed Mar 7**

*work work work work*

**example% Time to leave!**

*work work work work*

**example% You're going to be late!**

**FILES**

**.login**
**.profile**

**SEE ALSO**

**calendar**(1)

## NAME

lex – lexical analysis program generator

## SYNOPSIS

**lex** [ **–fntv** ] [ *filename* ] ...

## DESCRIPTION

**lex** generates programs to be used in simple lexical analysis of text. Each *filename* (the standard input by default) contains regular expressions to search for, and actions written in C to be executed when expressions are found.

A C source program, **lex.yy.c** is generated, to be compiled as follows:

**cc lex.yy.c –ll**

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized. The actual string matched is left in **yytext**, an external character array.

Matching is done in order of the strings in the file. The strings may contain square braces to indicate character classes, as in **[abx–z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators *, + and ?, which mean, respectively, any nonnegative number, any positive number, or either zero or one occurrences of the previous character or character-class. The "dot" character ('.') is the class of all ASCII characters except NEWLINE.

Parentheses for grouping and vertical bar for alternation are also supported. The notation $r\{d,e\}$ in a rule indicates instances of regular expression $r$ between $d$ and $e$. It has a higher precedence than |, but lower than that of *, ?, +, or concatenation. The ^ (carat character) at the beginning of an expression permits a successful match only immediately after a NEWLINE, and the $ character at the end of an expression requires a trailing NEWLINE.

The / character in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, although the remainder of the expression must follow in the input stream.

An operator character may be used as an ordinary symbol if it is within '"' symbols or preceded by '\'.

Three subroutines defined as macros are expected: **input()** to read a character; **unput**(*c*) to replace a character read; and **output**(*c*) to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex()**, and the library contains a **main()** which calls it. The action REJECT on the right side of the rule rejects this match and executes the next suitable match; the function **yymore()** accumulates additional characters into the same **yytext**; and the function **yyless**(*n*) where *n* is the number of characters to retain in **yytext**. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

In a **lex** program, any line beginning with a blank is assumed to contain only C text and is copied; if it precedes %% it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a %%, as in YACC. Lines preceding %% which begin with a nonblank character define the string on the left to be the remainder of the line; it can be used later by surrounding it with {}. Note: curly brackets do not imply parentheses; only string substitution is done.

The external names generated by **lex** all begin with the prefix **yy** or **YY**.

Certain table sizes for the resulting finite-state machine can be set in the definitions section:

**%p** *n* number of positions is *n* (default 2000)

**%n** *n* number of states is *n* (500)

**%t** *n* number of parse tree nodes is *n* (1000)

**%a** *n* number of transitions is *n* (3000)

The use of one or more of the above automatically implies the **–v** option, unless the **–n** option is used.

**OPTIONS**

**–f** Faster compilation. Do not bother to pack the resulting tables; limited to small programs.

**–n** Opposite of **–v**; **–n** is default.

**–t** Place the result on the standard output instead of in file **lex.yy.c**.

**–v** Print a one-line summary of statistics of the generated analyzer.

**EXAMPLES**

The following command line:

**lex lexcommands**

would draw **lex** instructions from the file **lexcommands**, and place the output in **lex.yy.c**.

The following:

```
%% [A–Z]   putchar (yytext[0]+'a'–'A'); [ ]+$     ; [ ]+      putchar(' ');
```

is an example of a **lex** program. It converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
D       [0–9]
%%
if      printf("IF statement\n");
[a–z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
                        {
                        case '/': break;
                        case '*': unput('*');
                        default: go to loop;
                        }
                }
```

**FILES**

**lex.yy.c**

**SEE ALSO**

**sed**(1V), **yacc**(1)

*Programming Utilities and Libraries*

**NOTES**

The **lex** command is not changed to support 8-bit symbol names, as this would produce **lex** source code that is not portable between systems.

**NAME**

line – read one line

**SYNOPSIS**

**line**

**DESCRIPTION**

**line** copies one line (up to a NEWLINE) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints a NEWLINE at least. It is often used within shell scripts to read a line from the terminal.

**SEE ALSO**

**sh**(1), **read**(2V)

## NAME

lint – a C program verifier

## SYNOPSIS

**lint** [ **–abchinquvxz** ] [ **–D***name* [*=def*] ] [ **–host=***arch* ] [ **–I***directory* ] [ **–l***library* ] [ **–o** *outputfile* ] [ **–target=***arch* ] [ **–U***name* ] *filename* ...

**lint** [ **–C***library* ] [ **–D***name* [*=def*] ] [ **–host=***arch* ] [ **–I***directory* ] [ **–l***library* ] [ **–target=***arch* ] [ **–U***name* ] *filename* ...

## SYSTEM V SYNOPSIS

**/usr/5bin/lint** [ **–abcghnpquvxzO** ] [ **–D***name* [*=def*] ] [ **–I***directory* ] [ **–l***library* ] [ **–o** *outputfile* ] [ **–U***name* ] *filename* ...

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**lint** attempts to detect features of the named C program files that are likely to be bugs, to be non-portable, or to be wasteful. It also performs stricter type checking than does the C compiler. **lint** runs the C preprocessor as its first phase, with the preprocessor symbol **lint** defined to allow certain questionable code to be altered or skipped by **lint**. Therefore, this symbol should be thought of as a reserved word for all code that is to be checked by **lint**.

Among the possible problems that are currently noted are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions with constant values. Function calls are checked for inconsistencies, such as calls to functions that return values in some places and not in others, functions called with varying numbers of arguments, function calls that pass arguments of a type other than the type the function expects to receive, functions whose values are not used, and calls to functions not returning values that use the non-existent return value of the function.

Filename arguments ending with **.c** are taken to be C source files. Filename arguments with names ending with **.ln** are taken to be the result of an earlier invocation of **lint**, with either the **–i** or the **–C** option in effect. The **.ln** files are analogous to the **.o** (object) files produced by the cc(1V) from **.c** files. **lint** also accepts special libraries specified with the **–l** option, which contain simplified definitions of standard library routines (preprocessed by '**lint –C**') for faster checking of function calls.

**lint** processes the various **.c**, **.ln**, and **llib-l***library***.ln** (lint library) files and process them in command-line order. By default, **lint** appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. When the **–C** and **–i** options are *omitted* the second pass of **lint** checks this list of files for mutual compatibility. When the **–C** or the **–i** options are used, the **.ln** and the **llib-l***library***.ln** files are ignored.

## SYSTEM V DESCRIPTION

Filename arguments with names ending with **.ln** are taken to be the result of an earlier invocation of **lint**, with either the **–c** or the **–o** option in effect.

**lint** processes the various **.c**, **.ln**, and **llib-l***library***.ln** (lint library) files and process them in command-line order. By default, **lint** appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **–p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **–c** option is *omitted* the second pass of **lint** checks this list of files for mutual compatibility. When the **–c** option is used, the **.ln** and the **llib-l***library***.ln** files are ignored.

**lint** produces its first-pass output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. If the **–c** option is not used, information gathered from all input files is then collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name is printed, followed by a question mark.

**OPTIONS**

**–a** Report assignments of **long** values to variables that are not **long**.

**–b** Report **break** statements that cannot be reached. This is not the default because, unfortunately, most **lex**(1) and many **yacc**(1) outputs produce many such complaints.

**–c** Complain about casts which have questionable portability.

**–h** Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.

**–i** Produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of **lint**'s first pass only, and are not checked for compatibility between functions.

**–n** Do not check compatibility against the standard library.

**–q** Do not complain about constructs that do not cause portability problems between current Sun implementations of the C language but that will cause portability problems between other implementations. If the **–q** flag is specified, **lint** treats type **enum** as an **int**, treats type **long** as type **int** and type **unsigned long** as **unsigned int**, and treats a 0 argument as being conformable with any pointer.

**–u** Do not complain about functions and external variables used and not defined, or defined and not used (this is suitable for running **lint** on a subset of files comprising part of a larger program).

**–v** Suppress complaints about unused arguments in functions.

**–x** Report variables referred to by **extern** declarations, but never used.

**–z** Do not complain about structures that are never defined (for example, using a structure pointer without knowing its contents).

**–C***library*
Create a **lint** library with the name **llib-l***library***.ln**.

**–D***name*[*=def*]
Define *name* for **cpp**(1), as if by a **#define** directive. If no definition is given, *name* is defined as 1.

**–host=***arch* (Sun-2, Sun-3, and Sun-4 systems only)
Define the host architecture to be *arch*. The default is the architecture returned by the **arch**(1) command. *arch* can be one of **sun2**, **sun3**, or **sun4**.

**–I***directory*
Add *directory* to the list of list of directories in which to search for include files. Include files with names that do not begin with / are always sought first in the directory of the *filename* argument, then in directories named in –I options, then in the **/usr/include** directory.

**–l***library*
Use the lint library *library* from the **/usr/lib/lint** directory.

**–o** *outputfile*
Name the output file *outputfile*. *outputfile* cannot be the same as *sourcefile* (**lint** will not overwrite the source file).

**–target=***arch* (Sun-2, Sun-3, and Sun-4 systems only)
Define the target architecture to be *arch*, for additional portability checks specific to that architecture. The default is the value returned by the **arch**(1) command. *arch* can be one of: **sun2**, **sun3**, or **sun4**.

**–U***name*
Remove any initial definition of *name* for the preprocessor.

**SYSTEM V OPTIONS**

The sense of the **–a**, **–b**, **–h**, and **–x** options is reversed in the System V version of **lint**; the tests they control *are* performed unless the flag is specified. The **–C** option is not available; instead, the **–c** or **–o** options can be used. The **–i** option is not used; instead, the **–c** option can be used. The **–q**, **–host**, and **–target** options are not available.

**–c** Produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of **lint**'s first pass only, and are not checked for compatibility between functions.

**–g**
**–O** These options are accepted but ignored. By recognizing these options, **lint**'s behavior is closer to that of the **cc**(1V) command.

**–n** Do not check compatibility against either the standard or the portable lint library.

**–p** Attempt to check portability of code to other dialects of C, such as IBM 370 and Honeywell GCOS. Along with performing stricter checking, this option truncates all non-external names to eight characters, and all external names to six characters and one case.

**–o** *library*
Create a lint library with the name **llib-l***lib***.ln**. The **–c** option nullifies any use of the **–o** option. The lint library produced is the input that is given to **lint**'s second pass. The **–o** option simply saves this file in the named lint library. To produce a **llib-l***lib***.ln** without extraneous messages, use of the **–x** option is suggested. The **–v** option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint comments" (see **Input Grammar** below).

**USAGE**

For more information about **lint** refer to **lint** in *Programming Utilities and Libraries*

To create lint libraries, use the **–C** option. For example

**example% lint –Ccongress** *filenames* ...

where *filenames* are the C sources of library *congress*, produces a file **llib-lcongress.ln** in the current directory in the correct library format suitable for "linting" programs using **–lcongress**.

**Input Grammar**

**lint**'s first pass reads standard C source files. **lint** recognizes the following C comments as commands.

**/*NOTREACHED*/**
At appropriate points, inhibit comments about unreachable code. (This comment is typically placed just after calls to functions like **exit**(2V)).

**/*VARARGS*n*/**
Suppress the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0. In this version of **lint**, **/*VARARGS0*/** is allowed. It no longer indicates the absence of variable arguments.

**/*ARGSUSED*/**
Enable the **–v** option for the next function.

**/*LINTLIBRARY*/**
At the beginning of a file, shut off complaints about unused functions and function arguments in this file. This is equivalent to using the **–v** and **–x** options.

## SYSTEM V USAGE

The behavior of the **–c** and the **–o** options allows for incremental use of **lint** on a set of C source files. Invoking '**lint –c**' for each source file produces a corresponding **.ln** file, and prints all messages pertaining to that source file. After all of the source files have been run through **lint** separately, it is invoked once more (without the **–c** option), and with all of the **.ln** files and **–lx** options. This produces messages about any inconsistencies between files. This scheme works well with **make**(1), since it allows **make** to "lint" only those source files that have been modified since the last time **lint** was run.

To create lint libraries, use the **–o** option. For example

**example% lint –x –o congress** *filenames* ...

where *filenames* are the C sources of library *congress*, produces a file **llib-lcongress.ln** in the current directory in the correct library format suitable for "linting" programs using **–lcongress**.

## EXAMPLE

The following **lint** call:

**example% lint –b myfile.c**

checks the consistency of the code in the C source file file **myfile.c**. The **–b** option indicates that unreachable **break** statements are to be checked for.

## FILES

| | |
|---|---|
| **/usr/lib/lint/lint[12]** | programs |
| **/usr/lib/lint/llib-l*.ln** | various prebuilt lint libraries |
| **/usr/lib/lint/llib-l*** | sources of the prebuilt lint libraries |

The following lint libraries are supplied with SunOS: **–lc**, **–lcore**, **–lcurses**, **–lkvm**, **–llwp**, **–lm**, **–lmp**, **–lpixrect**, **–lplot**, **–lsuntool**, **–lsunwindow**, **–ltermcap**, and **–ltermlib**. Additional lint libraries may be installed separately.

## SYSTEM V FILES

| | |
|---|---|
| **/usr/5lib/lint/lint[12]** | programs |
| **/usr/5lib/lint/llib-l*.ln** | various prebuilt lint libraries |
| **/usr/5lib/lint/llib-l*** | sources of the prebuilt lint libraries |

The following System V lint libraries are supplied with SunOS: **–lc**, **–lcore**, **–lcurses**, **–lkvm**, **–llwp**, **–lm**, **–lmp**, **–lpixrect**, **–lplot**, **–lport**. **–lsuntool**, and **–lsunwindow**. Additional lint libraries may be installed separately.

## SEE ALSO

**cc**(1V), **cpp**(1), **lex**(1), **make**(1), **yacc**(1), **exit**(2V), **setjmp**(3V), **ansic**(7V), **bsd**(7V), **posix**(7V), **sunos**(7), **svidii**(7V), **svidiii**(7V), **xopen**(7V)

*Programming Utilities and Libraries*

## NOTES

Because **cc** does not generate or support 8-bit symbol names, it is inappropriate to make **lint** 8-bit clean. See **cc**(1V) for an explanation about why **cc** is not 8-bit clean.

## BUGS

There are some things you just *cannot* get **lint** to shut up about.

The routines **exit**(2V), **longjmp** (see **setjmp**(3V)) and other functions that do not return are not understood; this causes various incorrect diagnostics.

Libraries created by the **–C** or **–o** options will, when used in later **lint** runs, cause certain errors that were reported when the libraries were created to be reported again, and cause line numbers and file names from the original source used to create those libraries to be reported in error messages. For these reasons, it is still useful to produce stripped down lint library source files and to use them to generate lint libraries.

## NAME

ln – make hard or symbolic links to files

## SYNOPSIS

**ln** [ **–fs** ] *filename* [ *linkname* ]
**ln** [ **–fs** ] *pathname... directory*

## SYSTEM V SYNOPSIS

**/usr/5bin/ln** [ **–fFs** ] *filename* [ *linkname* ]
**ln** [ **–fFs** ] *pathname... directory*

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**ln** creates an additional directory entry, called a *link*, to a file or directory. Any number of links can be assigned to a file. The number of links does not affect other file attributes such as size, protections, data, etc.

*filename* is the name of the original file or directory. *linkname* is the new name to associate with the file or filename. If *linkname* is omitted, the last component of *filename* is used as the name of the link.

If the last argument is the name of a directory, symbolic links are made in that directory for each *pathname* argument; **ln** uses the last component of each *pathname* as the name of each link in the named *directory*.

A hard link (the default) is a standard directory entry just like the one made when the file was created. Hard links can only be made to existing files. Hard links cannot be made across file systems (disk partitions, mounted file systems). To remove a file, all hard links to it must be removed, including the name by which it was first created; removing the last hard link releases the inode associated with the file.

A symbolic link, made with the –s option, is a special directory entry that points to another named file. Symbolic links can span file systems and point to directories. In fact, you can create a symbolic link that points to a file that is currently absent from the file system; removing the file that it points to does not affect or alter the symbolic link itself.

A symbolic link to a directory behaves differently than you might expect in certain cases. While an **ls**(1V) on such a link displays the files in the pointed-to directory, an '**ls –l**' displays information about the link itself:

```
example% ln –s dir link
example% ls link
file1 file2 file3 file4
example% ls –l link
lrwxrwxrwx  1 user        7 Jan 11 23:27 link -> dir
```

When you **cd**(1) to a directory through a symbolic link, you wind up in the pointed-to location within the file system. This means that the parent of the new working directory is not the parent of the symbolic link, but rather, the parent of the pointed-to directory. For instance, in the following case the final working directory is **/usr** and not **/home/user/linktest**.

```
example% pwd
/home/user/linktest
example% ln –s /usr/tmp symlink
example% cd symlink
example% cd ..
example% pwd
/usr
```

C shell user's can avoid any resulting navigation problems by using the **pushd** and **popd** built-in commands instead of **cd**.

**SYSTEM V DESCRIPTION**

The *System V* version of **ln** behaves as described above except for the following. If the linkname is an existing file and its mode does not forbid writing, then its contents are destroyed. If however its mode does not allow writing, the mode is printed, and the user asked for a response.

**OPTIONS**

**–f** Force a hard link to a directory — this option is only available to the super-user.

**–s** Create a symbolic link or links.

**SYSTEM V OPTIONS**

**–f** Force files to be linked without displaying permissions, asking questions or reporting errors.

**–F** Force a hard link to a directory — this option is only available to the super-user.

**–s** Create a symbolic link or links.

**EXAMPLE**

The commands below illustrate the effects of the different forms of the **ln** command:

```
example% ln file link
example% ls –F file link
file  link
example% ln –s file symlink
example% ls –F file symlink
file  symlink@
example% ls –li file link symlink
 10606 -rw-r--r--  2 user         0 Jan 12 00:06 file
 10606 -rw-r--r--  2 user         0 Jan 12 00:06 link
 10607 lrwxrwxrwx  1 user         4 Jan 12 00:06 symlink -> file
example% ln –s nonesuch devoid
example% ls –F devoid
devoid@
example% cat devoid
devoid: No such file or directory
example% ln –s /proto/bin/* /tmp/bin
example% ls –F /proto/bin /tmp/bin
/proto/bin:
x*    y*    z*

/tmp/bin:
x@    y@    z@
```

**SEE ALSO**

**cp**(1), **ls**(1V), **mv**(1), **rm**(1), **link**(2V), **readlink**(2), **stat**(2V), **symlink**(2)

**BUGS**

When the last argument is a directory, simple basenames should not be used for *pathname* arguments. If a basename is used, the resulting symbolic link points to itself:

```
example% ln –s file /tmp
example% ls –l /tmp/file
lrwxrwxrwx  1 user         4 Jan 12 00:16 /tmp/file -> file
example% cat /tmp/file
/tmp/file: Too many levels of symbolic links
```

To avoid this problem, use full pathnames, or prepend a reference to the **PWD** variable to files in the working directory:

```
example% rm /tmp/file
example% ln -s $PWD/file /tmp
lrwxrwxrwx  1 user        4 Jan 12 00:16 /tmp/file -> /home/user/subdir/file
```

## NAME

load, loadc – load Application SunOS or Developer's Toolkit clusters

## SYNOPSIS

**load** [ *filename* ... ]

**loadc** [ *cluster* ... ]

**loadc appl**

**loadc devel**

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

**load** loads the optional clusters in the Application SunOS or the Developer's Toolkit that contain the files specified in the *filename* arguments. **loadc** loads the optional clusters in the Application SunOS or the Developers Toolkit specified in the cluster arguments. Specify the special cluster name **appl** with **loadc** to load all the Application SunOS clusters. Likewise, specify **devel** to **loadc** to load all the Developer's Toolkit clusters.

**load** and **loadc** require the user to specify the distribution medium, either 3.5-inch diskette or quarter-inch tape, and insert the diskette or tape. The user will be asked to confirm that the specified medium has been inserted. If the user confirmation is negative, no software will be loaded.

Without arguments, **load** and **loadc** display a summary of the clusters in the Application SunOS and Developer's Toolkit, including the load state and size of each cluster.

## EXAMPLES

To load the cluster that contains the **spell**(1) command:

```
example% load spell
Enter your distribution media type (1=1/4" tape, 2=3.5" diskette): 2
Insert diskette n to load the spellcheck cluster, confirm (y/n): y
Loading the spellcheck cluster ...
The spellcheck cluster has been loaded.
space used by clusters: 6021K bytes
total space remaining: 20432K bytes
example%
```

To load the **spellcheck** cluster:

```
example% loadc spellcheck
Enter your distribution media type (1=1/4" tape, 2=3.5" diskette): 2
Insert diskette n to load the spellcheck, confirm (y/n): y
Loading the spellcheck cluster ...
The spellcheck cluster has been loaded.
space used by clusters: 6021K bytes
total space remaining: 20432K bytes
example%
```

To display a summary of the clusters in the Application SunOS and Developer's Toolkit:

```
example% load
Application SunOS Optional Clusters:
  available  cluster          size (bytes)
  ---------  -------          ----
    yes      accounting       265K
    no       advanced_admin   501K
    ...
```

```
Developer's Toolkit Optional Clusters:
  available  cluster        size (bytes)
  ---------  -------        ----
   no        base_devel     6907K
   ...

space used by clusters: 6021K bytes
total space remaining: 20432K bytes
example%
```

A cluster is available if it has been "loaded" using **load** or **loadc** or if it has been "mounted" across the network.

## ENVIRONMENT

**LOADMEDIA** Used to specify the distribution media type for the system. It can be set to **diskette** to specify 3.5-inch diskette or **tape** to specify 1/4" tape. If it is set, **load** and **loadc** will not ask the user to enter the distribution media type.

## FILES

| | |
|---|---|
| **/usr/cluster/appl** | where Application SunOS clusters are loaded (or mounted) |
| **/usr/cluster/devel** | where Developer's Toolkit clusters are loaded (or mounted) |
| **/usr/lib/load/*** | data files |

## SEE ALSO

**cluster(1), unload**(1), **toc**(5)

*Sun386i System Setup and Maintenance*

## DIAGNOSTICS

**Wrong** *diskette/tape*
An incorrect diskette or tape was inserted. The user will again be asked to insert the specified media.

**The file** *filename* **is not in any of the optional software clusters.**
The specified file is not part of the Application SunOS or Developer's Toolkit.

**There is no** *cluster* **cluster.**
The specified cluster is not part of the Application SunOS or Developers Toolkit.

**The cluster** *cluster* **is already loaded, overwrite? (y/n):**
The specified cluster appears to have been loaded already. Type **y** followed by RETURN to have the cluster loaded or **n** followed by RETURN to cancel the loading of the cluster.

**Cluster** *cluster* **requires** *n***K; there is not enough disk space.**
There is not enough disk space to hold the specified cluster.

**The** *cluster* **cluster has not been loaded.**
The loading of the specified cluster has been canceled or interrupted by the user.

**The Application SunOS (and/or) Developers Toolkit are mounted.**
The Application SunOS or Developers Toolkit or both are mounted across the network and can not be loaded or unloaded.

**The** *tape/diskette* **drive is currently in use.**
You are trying to load a cluster from tape (or diskette) and another process currently has control of the tape (or diskette) drive.

## NAME

loadkeys, dumpkeys – load and dump keyboard translation tables

## SYNOPSIS

**loadkeys** [ **–e** ] [ *filename* ]

**dumpkeys**

## DESCRIPTION

### loadkeys

**loadkeys** reads the file specified by *filename*, and modifies the keyboard streams module's translation tables. If no file is specified, and the keyboard is a Type-4 keyboard, a default file for the layout indicated by the DIP switches on the keyboard. The file is in the format specified by **keytables**(5).

If the layout code in the DIP switches on the keyboard has the hexadecimal value **0x***dd*, the file loaded by **loadkeys** by default is **/usr/share/lib/keytables/layout_***dd*. These files specify only the entries that change between the different Type-4 keyboard layouts.

### dumpkeys

**dumpkeys** writes, to the standard output, the current contents of the keyboard streams module's translation tables, in the format specified by **keytables**(5).

## OPTIONS

**–e** **loadkeys** loads the requested keytable layout information into the EEPROM keytables. This allows the console monitor to select the correct key-values when a non-US keyboard is connected to the workstation.

## FILES

**/usr/share/lib/keytables/layout_***dd*
default keytable files

## SEE ALSO

**kb**(4M), **keytables**(5)

## BUGS

The **–e** option requires EEPROM rev 2.6 or later.

## NAME

lockscreen, lockscreen_default – maintain SunView context and prevent unauthorized access

## SYNOPSIS

**lockscreen** [ **–enr** ] [ **–b** *program* ] [ **–t** *seconds* ] [ *gfx-program* [ *gfx-program-arguments* ] ]

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information about installing optional software.

## DESCRIPTION

**lockscreen** is a SunView utility that locks the screen against unauthorized access while preserving the state of the SunView display. It clears the workstation screen to black, and then runs *gfx-program*, which typically provides a moving graphics display to reduce phosphor burn. When no *gfx-program* is provided, a suitable default program is run.

**lockscreen** requires the user's password before restoring the window context. When any keyboard or mouse button is pressed, the graphics screen is replaced by a password screen that displays the user name, a small box with a bouncing logo, and a prompt for the user's password. If the user has no password, or if the **–n** option is used, the window context is simply restored.

When the password screen appears:

- Restore the window context by entering the user's password followed by a RETURN (this password is not echoed on the screen) or,
- Point to the black box and click the left button to return to the graphics display.

If neither of the above actions is taken, *gfx_program* will resume execution after the interval specified with the **–t** option.

## OPTIONS

**–e** Add the **Exit Desktop** choice to the password screen. If pointed to and clicked, the SunView environment is exited and the current user is logged out.

**–n** Require no password to reenter the window environment.

**–r** Allow the use of the user name **root** in the **'Name:'** field of the password screen. Normally, **root** is not accepted as a valid user name.

**–b** *program*
Allow an additional program to be run as a child process of **lockscreen**. This background process could be a compile server or some other useful program that the user wants run while lockscreen is running. No arguments are passed to this program.

**–t** *seconds*
After *seconds* seconds, clear the password screen and restart *gfx-program*. The default is 5 minutes (300 seconds).

[ *gfx-program* ] [ *gfx-program-arguments* ]
Run this program after clearing the screen to black. When no *program* argument is present, **lockscreen** tries to run **lockscreen_default** if it is in the standard search path; otherwise, a bouncing Sun logo appears. When *gfx-program-arguments* are specified and the *gfx-program* is not, the arguments are passed to **lockscreen_default**. **lockscreen_default** is typically a non-interactive graphics program (see **graphics_demos**(6)). **lockscreen** does not search for **lockscreen_default** when the *gfx-program* is specified explicitly as an empty argument (an adjacent pair of quotemarks).

## FILES

**/usr/bin/lockscreen_default**
Default *gfx-program*. This displays a series of **life**(6) patterns. If a file named **lockscreen_default** appears earlier in the search path, that file is used instead.

**SEE ALSO**

**login**(1), **sunview**(1)

**BUGS**

**lockscreen** is more secure when *gfx-program* are not specified and when **lockscreen_default** is not in the search path. To improve security, enter the following command as root, or start **lockscreen** with an empty argument (see OPTIONS, above).

**# mv /usr/bin/lockscreen_default /usr/bin/lockscreen_default-**

## NAME

logger – add entries to the system log

## SYNOPSIS

**logger** [ **–t** *tag* ] [ **–p** *priority* ] [ **–i** ] [ **–f** *filename* ] [ *message* ] ...

## DESCRIPTION

**logger** provides a method for adding one-line entries to the system log file from the command line. One or more *message* arguments can be given on the command line, in which case each is logged immediately. Otherwise, a *filename* can be specified, in which case each line in the file is logged. If neither is specified, **logger** reads and logs messages on a line-by-line basis from the standard input.

## OPTIONS

| | |
|---|---|
| **–t** *tag* | Mark each line added to the log with the specified *tag*. |
| **–p** *priority* | Enter the message with the specified *priority*. The message priority can be specified numerically, or as a *facility.level* pair. For example, '**–p local3.info**' assigns the message priority to the **info** level in the **local3** facility. The default priority is **user.notice**. |
| **–i** | Log the process ID of the **logger** process with each line. |
| **–f** *filename* | Use the contents of *filename* as the message to log. |
| *message* | If this is unspecified, either the file indicated with **–f** or the standard input is added to the log. |

## EXAMPLES

**logger System rebooted**

will log the message '**System rebooted**' to the facility at priority **notice** to be treated by **syslogd** as other messages to the facility **notice** are.

**logger –p local0.notice –t HOSTIDM –f /dev/idmc**

will read from the file **/dev/idmc** and will log each line in that file as a message with the tag '**HOSTIDM**' at priority **notice** to be treated by **syslogd** as other messages to the facility **local0** are.

## SEE ALSO

**syslog**(3), **syslogd**(8)

## NAME

login – log in to the system

## SYNOPSIS

**login** [ **–p** ] [ *username* ]

## DESCRIPTION

**login** signs *username* on to the system initially; **login** may also be used at any time to change from one user ID to another.

When used with no argument, **login** requests a user name and password (if appropriate). Echoing is turned off (if possible) while typing the password. Note: the number of significant characters in a password is 8. See **passwd**(1).

If password aging is enabled and the password corresponding to the user name has expired, the user will be prompted for a new password. The user has to successfully modify his password for login to proceed. See **passwd**(1).

When successful, **login** updates accounting files, prints disk usage and limits (by running **quota**(1),) prints the message of the day, informs you of the existence of any mail, and displays the time you last logged in. None of these messages is printed if there is a **.hushlogin** file in your home directory. This is mostly used to make life easier for nonhuman users, such as **uucp**(1C).

**login** initializes the user and group IDs and the working directory, then starts a command interpreter shell (usually either **sh**(1) or **csh**(1)) according to specifications found in the file **/etc/passwd**. Argument 0 of the command interpreter is the name of the command interpreter with a leading dash ('–') prepended.

**login** also modifies the environment (**environ**(5V)) with information specifying home directory, command interpreter, terminal-type (if available) and username. The **–p** argument preserves the remainder of the environment, otherwise any previous environment is discarded.

The super-user **root** may only log in on those terminals marked as "secure" in the **/etc/ttytab** file. Otherwise, the super-user must log in as an ordinary user and become super-user using **su**(1v). For example, if the file contained:

```
console  "/etc/getty Console-9600"      sun     on secure
tty00    "/etc/getty Console-9600"      sun     on
...
```

the super-user could only log in directly on the console. See **ttytab**(5) for a discussion of "secure" and other **getty**(8) options used in **/etc/ttytab**.

If the file **/etc/nologin** exists, **login** prints its contents on the user's terminal and exits. This is used by **shutdown**(8) to stop logins when the system is about to go down.

The **login** command, recognized by **sh**(1) and **csh**(1), is executed directly (without forking), and terminates that shell. To resume working, you must log in again.

**login** times out and exits if its prompt for input is not answered within a reasonable time.

When the Bourne shell (**sh**) starts up, it reads a file called **.profile** from your home directory (that of the username you use to log in). When the C shell (**csh**) starts up, it reads a file called **.cshrc** from your home directory, and then reads a file called **.login**.

The shells read these files only if they are owned by the person logging in.

## OPTIONS

**–p** Preserve any existing environment variables and their values; otherwise the previous environment is discarded.

## FILES

| | |
|---|---|
| **/etc/utmp** | accounting |
| **/var/adm/wtmp** | accounting |
| **/var/adm/lastlog** | time of last login |

| | |
|---|---|
| **/etc/ttytab** | terminal types |
| **/usr/ucb/quota** | quota check |
| **/var/spool/mail/*** | mail |
| **/etc/motd** | message-of-the-day |
| **/etc/passwd** | password file |
| **/etc/nologin** | stop login, print message |
| **~/.hushlogin** | makes login quieter |
| **.login** | |
| **.profile** | |

SEE ALSO

**csh**(1), **mail**(1), **passwd**(1), **quota**(1), **rlogin**(1C), **sh**(1), **uucp**(1C), **environ**(5V), **fbtab**(5), **passwd**(5), **svdtab**(5), **utmp**(5V), **getty**(8), **init**(8), **shutdown**(8)

DIAGNOSTICS

**Login incorrect** If the name or the password is bad (or mistyped).

**No Shell**

**cannot open password file**

**no directory** Ask your system administrator for assistance.

NOTES

The following options are undocumented, and not intended for the user. The **–r** option is used by the remote login server, **rlogind**(8C) to force **login** to enter into an initial connection protocol. **–h** is used by **telnetd**(8C) and other servers to list the host from which the connection was received.

The following warnings apply when **login** account names contain characters outside the range of 7-bit ASCII:

- A user cannot **rlogin** to *account-name* containing 8-bit characters from a system that does not handle 8-bit characters.
- A user cannot log in to *account-name* containing 8-bit characters from a 7-bit ASCII terminal (through a modem, for example).
- Several parts of the C library are not tested for 8-bit user names.
- Codeset mapping can vary between systems, so an 8-bit pattern can represent different characters on different sytems.
- Password algorithms do not work with 8-bit characters in the password.

**NAME**

logname – get the name by which you logged in

**SYNOPSIS**

**logname**

**DESCRIPTION**

**logname** returns the contents of the environment variable LOGNAME, which is set when a user logs into the system.

**FILES**

**/etc/profile**

**SEE ALSO**

**env**(1), **login**(1), **environ**(5V)

**NOTES**

The following warnings apply when **login** accounts contain characters outside the range of 7-bit ASCII:

- A user cannot **rlogin** to *account-name* containing 8-bit characters from a system that does not handle 8-bit characters.
- A user cannot log in to *account-name* containing 8-bit characters from a 7-bit ASCII terminal (through a modem, for example).
- Several parts of the C library are not tested for 8-bit user names.
- Codeset mapping can vary between systems, so an 8-bit pattern can represent different characters on different sytems.
- Password algorithms do not work with 8-bit characters.

**NAME**

look – find words in the system dictionary or lines in a sorted list

**SYNOPSIS**

**look** [ **–df** ] [ **–t***c* ] *string* [ *filename* ]

**DESCRIPTION**

**look** consults a sorted *filename* and prints all lines that begin with *string*.

If no *filename* is specified, **look** uses **/usr/dict/words** with collating sequence **–df**.

**OPTIONS**

**–d** Dictionary order. Only letters, digits, TAB and SPACE characters are used in comparisons.

**–f** Fold case. Upper case letters aren't distinguished from lower case in comparisons.

**–t***c* Set termination character. All characters to the right of *c* in *string* are ignored.

**FILES**

**/usr/dict/words**

**SEE ALSO**

**grep**(1V), **sort**(1V)

## NAME

lookbib – find references in a bibliographic database

## SYNOPSIS

**lookbib** *database*

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a '**%**', followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with '**%**'.

**lookbib** uses an inverted index made by **indxbib** to find sets of bibliographic references. It reads keywords typed after the '**>**' prompt on the terminal, and retrieves records containing all these keywords. If nothing matches, nothing is returned except another '**>**' prompt.

It is possible to search multiple databases, as long as they have a common index made by **indxbib**(1). In that case, only the first argument given to **indxbib** is specified to **lookbib.**

If **lookbib** does not find the index files (the **.i[abc]** files), it looks for a reference file with the same name as the argument, without the suffixes. It creates a file with a **.ig** suffix, suitable for use with **fgrep** (see **grep**(1V)). **lookbib** then uses this **fgrep** file to find references. This method is simpler to use, but the **.ig** file is slower to use than the **.i[abc]** files, and does not allow the use of multiple reference files.

## FILES

| | |
|---|---|
| *x*.**ia** | |
| *x*.**ib** | |
| *x*.**ic** | index files |
| *x*.ig | reference file |

## SEE ALSO

**addbib**(1), **grep**(1V), **indxbib**(1), **refer**(1), **roffbib**(1), **sortbib**(1)

*Formatting Documents*

## BUGS

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

**NAME**

lorder – find an ordering relation for an object library

**SYNOPSIS**

**lorder** *filename*...

**DESCRIPTION**

Give **lorder** one or more object or library archive (see **ar**(1V)) *filename*s, and it lists pairs of object file names — meaning that the first file of the pair refers to external identifiers defined in the second — to the standard output. **lorder**'s output may be processed by **tsort**(1) to find an ordering of a library suitable for one-pass access by **ld**(1).

**EXAMPLE**

This brash one-liner intends to build a new library from existing **.o** files.

```
ar cr library `lorder *.o | tsort`
```

The **ranlib**(1), command converts an ordered archive into a randomly accessed library and makes **lorder** unnecessary.

**SEE ALSO**

**ar**(1V), **ld**(1), **ranlib**(1), **tsort**(1)

**BUGS**

The names of object files, in and out of libraries, must end with '**.o**'; otherwise, nonsense results.

## NAME

lp, cancel – send/cancel requests to a printer

## SYNOPSIS

**lp** [ **–cmsw** ] [ **–d***dest* ] [ **–n***number* ] [ **–o***option* ] [ **–t***title* ] *filename*...

**cancel** [ *ids* ] [ *printers* ]

## DESCRIPTION

**lp** arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name '–' stands for the standard input and may be supplied on the command line in conjunction with named *filenames*. The order in which *filenames* appear is the same order in which they will be printed.

**lp** associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see **cancel**) or find the status (see **lpstat**(1)) of the request.

**cancel** cancels line printer requests. The command line arguments may be either request *ids* (as returned by **lp**(1)) or *printer* names. When *ids* are specified, **cancel** removes the jobs corresponding to the *ids* on the destination printer. See the **–d** option for details on how the default destination printer is obtained. For a complete list of printers, use **lpstat**(1). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

## OPTIONS

The following options to **lp** may appear in any order and may be intermixed with file names:

**–c** Make copies of the *filenames* to be printed immediately when **lp** is invoked. Otherwise, *filenames* will not be copied, but will be linked whenever possible. If the **–c** option is not given, then the user should be careful not to remove any of the *filenames* before the request has been printed in its entirety. It should also be noted that in the absence of the **–c** option, any changes made to the named *filenames* after the request is made but before it is printed will be reflected in the printed output.

**–m** Send mail (see **mail**(1)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

**–s** Suppress messages from *lp*(1) such as '**request id is ...**'.

**–w** Write a message on the user's terminal after the *filenames* have been printed. If the user is not logged in locally, then mail will be sent instead.

**–d***dest* Choose *dest* as the printer that is to do the printing. By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, *dest* will be taken from the environment variable **PRINTER**. If **PRINTER** is not defined either, then the default destination for the computer system is used. Destination names vary between systems (see **lpstat**(1)).

**–n***number* Print *number* copies (default of 1) of the output.

**–o***option* Specify printer-dependent *options*. Several such options may be collected by specifying the **–o** option more than once.

**–t***title* Print *title* on the banner page of the output.

## FILES

**/etc/termcap**

## SEE ALSO

**lpstat**(1), **lpr**(1), **lprm**(1), **mail**(1), **lpc**(8), **lpd**(8)

## NAME

lpq – display the queue of printer jobs

## SYNOPSIS

**lpq** [ **–P***printer* ] [ **–l** ] [ + [ *interval* ] ] [ *job#* ... ] [ *username* ... ]

## DESCRIPTION

**lpq** displays the contents of a printer queue. It reports the status of jobs specified by *job#*, or all jobs owned by the user specified by *username*. **lpq** reports on all jobs in the default printer queue when invoked with no arguments.

For each print job in the queue, **lpq** reports the user's name, current position, the names of input files comprising the job, the job number (by which it is referred to when using **lprm**(1)) and the total size in bytes. Normally, only as much information as will fit on one line is displayed. Jobs are normally queued on a first-in-first-out basis. Filenames comprising a job may be unavailable, such as when **lpr** is used at the end of a pipeline; in such cases the filename field indicates "(standard input)".

If **lpq** warns that there is no daemon present (that is, due to some malfunction), the **lpc**(8) command can be used to restart a printer daemon.

## OPTIONS

**–P** *printer* — Display information about the queue for the specified *printer*. In the absence of the **–P** option, the queue to the printer specified by the PRINTER variable in the environment is used. If the PRINTER variable isn't set, the queue for the default printer is used.

**–l** — Display queue information in long format; includes the name of the host from which the job originated.

+[ *interval* ] — Display the spool queue periodically until it empties. This option clears the terminal screen before reporting on the queue. If an *interval* is supplied, **lpq** sleeps that number of seconds in between reports.

## FILES

| | |
|---|---|
| **/etc/termcap** | for manipulating the screen for repeated display |
| **/etc/printcap** | to determine printer characteristics |
| **/var/spool/l*** | spooling directory, as determined from printcap |
| **/var/spool/l*/cf*** | control files specifying jobs |
| **/var/spool/l*/lock** | lock file to obtain the currently active job |

## SEE ALSO

**lpr**(1), **lprm**(1), **lpc**(8), **lpd**(8)

## DIAGNOSTICS

*printer* **is ready and printing**

The **lpq** program checks to see if there is a printer daemon. If the daemon is hung, the super-user can abort the current daemon and start a new one using **lpc**(8).

**Waiting for** *printer* **to become ready (offline ?)**

The daemon could not open the printer device. The printer may be turned off-line. This message can also occur if a printer is out of paper, the paper is jammed, and so on. Another possible cause is that a process, such as an output filter, has exclusive use of the device. The only recourse in this case is to kill the offending process and restart the printer with **lpc**.

**waiting for** *host* **to come up**

A daemon is trying to connect to the remote machine named *host*, in order to send the files in the local queue. If the remote machine is up, **lpd** on the remote machine is probably dead or hung and should be restarted using *lpc*.

**sending to** *host*

The files are being transferred to the remote *host*, or else the local daemon has hung while trying to transfer the files.

**Warning:** *printer* **is down**

The printer has been marked as being unavailable with **lpc**.

**Warning: no daemon present**

The **lpd** process overseeing the spooling queue, as indicated in the "lock" file in that directory, does not exist. This normally occurs only when the daemon has unexpectedly died. Check the printer's error log for a diagnostic from the deceased process; you can restart the printer daemon with **lpc**.

## BUGS

**lpq** may report unreliably. The status as reported may not always reflect the actual state of the printer. Under some circumstances, **lpq** reports that a printer is ready and printing when the daemon is, in fact, hung.

Output formatting is sensitive to the line length of the terminal; this can result in widely-spaced columns.

**lpq** is sometimes unable to open various files when the lock file is malformed.

## NAME

lpr – send a job to the printer

## SYNOPSIS

**lpr** [ **–P***printer* ] [ **–#***copies* ] [ **–C***class* ] [ **–J***job* ] [ **–T***title* ] [ **–i** [ *indent* ] ] [ **–1234***font* ]
[ **–w***cols* ] [ **–r** ] [ **–m** ] [ **–h** ] [ **–s** ] [ *–filter-option* ] [ *filename ...* ]

## DESCRIPTION

**lpr** creates a printer job in a spooling area for subsequent printing as facilities become available. Each printer job consists of a control file and one or more data files. The data files are copies of (or, with **–s** , symbolic links to) each *filename* you specify. The spool area is managed by the line printer daemon, **lpd**(8). Jobs that specify a printer on a remote machine are forwarded by **lpd**.

**lpr** reads from the standard input if no files are specified.

## OPTIONS

**–P***printer*
Send output to the named *printer*. Otherwise send output to the printer named in the **PRINTER** environment variable, or to the default printer, **lp**.

**–#***copies*
Produce the number of *copies* indicated for each named file. For example:

**example% lpr –#3 index.c lookup.c**

produces three copies of **index.c**, followed by three copies of **lookup.c**. On the other hand,

**example% cat index.c lookup.c | lpr –#3**

generates three copies of the concatenation of the files.

**–C***class*
Print *class* as the job classification on the burst page. For example,

**example% lpr –C Operations new.index.c**

replaces the system name (the name returned by *hostname*) with "Operations" on the burst page, and prints the file *new.index.c*.

**–J***job*
Print *job* as the job name on the burst page. Normally, **lpr** uses the first file's name.

**–T***title*
Use *title* instead of the file name for the title used by **pr**(1V).

**–i**[ *indent* ]
Indent output *indent* SPACE characters. Eight SPACE characters is the default. The indent is passed to the input filter. If no input filter is present, this option is ignored.

**–1***font*
**–2***font*
**–3***font*
**–4***font*
Mount the specified *font* on font position **1**, **2**, **3** or **4**. The daemon will construct a *.railmag* file in the spool directory that indicates the mount by referencing **/usr/lib/vfont/font.**

**–w***cols*
Use *cols* as the page width for **pr**.

**–r**
Remove the file upon completion of spooling, or upon completion of printing with the **–s** option.

**–m**
Send mail upon completion.

**–h**
Suppress printing the burst page.

**–s**
Create a symbolic link from the spool area to the data files rather than trying to copy them (so large files can be printed). This means the data files should not be modified or removed until they have been printed. This option can be used to avoid truncating files larger than the maximum given in the **mx** capability of the **printcap**(5) entry. **–s** only prevents copies of local files from being made. Jobs from remote hosts are copied anyway. **–s** only works with named data files; if the **lpr** command is at the end of a pipeline, the data is copied to the spool.

*filter-option* The following single letter options notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

**–p** Use **pr** to format the files (**lpr –p** is very much like '**pr | lpr**').
**–l** Print control characters and suppress page breaks.
**–t** The files contain **troff**(1) (cat phototypesetter) binary data.
**–n** The files contain data from **ditroff** (device independent troff).
**–d** The files contain data from **tex** (DVI format from Stanford).
**–g** The files contain standard plot data as produced by the **plot**(3X) routines (see also **plot**(1G) for the filters used by the printer spooler).
**–v** The files contain a raster image, see **rasterfile**(5). The printer must support an appropriate imaging model such as PostScript in order to print the image.
**–c** The files contain data produced by **cifplot**.
**–f** Interpret the first character of each line as a standard FORTRAN carriage control character.

If no *filter-option* is given (and the printer can interpret PostScript), the string '**%!**' as the first two characters of a file indicates that it contains PostScript commands.

These filter options offer a standard user interface, and all options may not be available for, nor applicable to, all printers.

## FILES

| | |
|---|---|
| **/etc/passwd** | personal identification |
| **/etc/printcap** | printer capabilities data base |
| **/usr/lib/lpd** | line printer daemon |
| **/var/spool/l*** | directories used for spooling |
| **/var/spool/l*/cf*** | daemon control files |
| **/var/spool/l*/df*** | data files specified in '**cf**' files |
| **/var/spool/l*/tf*** | temporary copies of '**cf**' files |
| **/usr/lib/vfont/font** | |

## SEE ALSO

**lpq**(1), **lprm**(1), **plot**(1G), **pr**(1V), **screendump**(1), **troff**(1), **plot**(3X), **printcap**(5), **rasterfile**(5), **lpc**(8), **lpd**(8)

## DIAGNOSTICS

**lpr: copy file is too large**
A file is determined to be too "large" to print by copying into the spool area. **lpr** truncates the file, and prints as much of it as it can. The maximum file length is specified by the **mx** capability of the **printcap**(5) entry for the printer. If no **mx** capability is specified, the default limit is 1000 Kbytes. Use the **–s** option as defined above to make a symbolic link to the file instead of copying it.

**lpr:** *printer*: **unknown printer**
The **printer** was not found in the **printcap** database. Usually this is a typing mistake; however, it may indicate a missing or incorrect entry in the **/etc/printcap** file.

**lpr:** *printer*: **jobs queued, but cannot start daemon.**
The connection to **lpd** on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket **/dev/printer** to be sure it still exists (if it does not exist, there is no **lpd** process running).

**lpr:** *printer*: **printer queue is disabled**

This means the queue was turned off with

**example% /usr/etc/lpc disable** *printer*

to prevent **lpr** from putting files in the queue. This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by a super-user with **lpc**.

If a connection to **lpd** on the local machine cannot be made **lpr** will say that the daemon cannot be started. Diagnostics may be printed in the daemon log file regarding missing spool files by **lpd**.

## BUGS

Command-line options cannot be combined into a single argument as with some other commands. The command:

**lpr –fs**

is not equivalent to

**lpr –f –s**

Placing the –s flag first, or writing each option as a separate argument, makes a link as expected.

**lpr –p** is not precisely equivalent to **pr | lpr**. **lpr –p** puts the current date at the top of each page, rather than the date last modified.

Fonts for **troff**(1) and $T_EX$® reside on the printer host. It is currently not possible to use local font libraries.

**lpr** refuses to print a.out files and library archives.

The –s option only avoids copying the data file to the spool directory of the local machine. If the printer for a job resides on a remote machine, the data file will be copied to the remote spool directory in all cases.

## NAME

lprm – remove jobs from the printer queue

## SYNOPSIS

**lprm** [ **–P***printer* ] [ – ] [ *job #* ... ] [ *username* ... ]

## DESCRIPTION

**lprm** removes a job or jobs from a printer's spooling queue. Since the spool directory is protected from users, using **lprm** is normally the only method by which a user can remove a job.

Without any arguments, **lprm** deletes the job that is currently active, provided that the user who invoked **lprm** owns that job.

When the super-user specifies a *username*, **lprm** removes all jobs belonging to that user.

You can remove a specific job by supplying its job number as an argument, which you can obtain using **lpq**(1). For example:

```
example% lpq -Phost
host is ready and printing
Rank   Owner  Job  Files                 Total Size
active wendy  385  standard input        35501 bytes
example% lprm -Phost 385
```

**lprm** reports the names of any files it removes, and is silent if there are no applicable jobs to remove.

**lprm** kills the active printer daemon, if necessary, before removing spooled jobs; it restarts the daemon when through.

## OPTIONS

| | |
|---|---|
| **–P***printer* | Specify the queue associated with a specific printer. Otherwise the value of the **PRINTER** variable in the environment is used. If this variable is unset, the queue for the default printer is used. |
| – | Remove all jobs owned by you. If invoked by the super-user, all jobs in the spool are removed. (Job ownership is determined by the user's login name and host name on the machine where the **lpr** command was invoked). |

## FILES

| | |
|---|---|
| **/etc/printcap** | printer characteristics file |
| **/var/spool/*** | spooling directories |
| **/var/spool/l*/lock** | lock file used to obtain the pid of the current daemon and the job number of the currently active job |

## SEE ALSO

**lpr**(1), **lpq**(1), **lpd**(8)

## DIAGNOSTICS

**lprm:** *printer*: **cannot restart printer daemon**

The connection to **lpd** on the local machine failed. This usually means the printer server started at boot time has died or is hung. If it is hung, the master **lpd**(8) daemon may have to be killed and a new one started.

## BUGS

Since race conditions are possible when updating the lock file, an active job may be incorrectly identified for removal by an **lprm** command issued with no arguments. During the interval between an **lpq**(1) command and the execution of **lprm**, the next job in line may have become active; that job may be removed unintentionally if it is owned by you. To avoid this, supply **lprm** with the job number to remove when a critical job that you own is next in line.

Only the super-user can remove print jobs submitted from another host.

## NAME

lpstat – display the printer status information

## SYNOPSIS

**lpstat** [ **–d** ] [ **–r** ] [ **–s** ] [ **–t** ] [ **–a** [ *list* ] ] [ **–c** [ *list* ] ] [ **–o** [ *list* ] ] [ **–p** [ *list* ] ] [ **–u** [ *list* ] ] [ **–v** [ *list* ] ]

## DESCRIPTION

**lpstat** prints information about the current status of the printer spooling system.

If no *options* are given, then **lpstat** prints the status of all requests made to **lp**(1) by the user. When the destination printer is not explicitly specified, the value of the **LPDEST** environment variable is used by default. If **LPDEST** is not defined, then the value of the **PRINTER** environment variable is used. If **PRINTER** is not defined either, then the system default printer is used. Any arguments that are not options are assumed to be request *ids* (as returned by **lp**). **lpstat** prints the status of such requests.

## OPTIONS

The options below may appear in any order and may be repeated and intermixed with other arguments. Some of the options below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more SPACE characters. For example:

**–u"user1, user2, user3"**

The omission of a *list* following such options prints all information relevant to the option, for example:

**lpstat –o**

prints the status of all output requests.

**–d** Print the system default destination for output requests.

**–r** Print the status of the printer request scheduler

**–s** Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

**–t** Print all status information.

**–a**[ *list* ] Print acceptance status of destinations for output requests. *list* is a list of intermixed printer names.

**–c**[ *list* ] Print class names and their members. *list* is a list of class names.

**–o**[ *list* ] Print the status of output requests. *list* is a list of intermixed printer names, class names, and request *ids*.

**–p**[ *list* ] Print the status of printers. *list* is a list of printer names.

**–u**[ *list* ] Print status of output requests for users on the default printer. *list* is a list of login names.

**–v**[ *list* ] Print the names of printers and the path names of the devices associated with them. *list* is a list of printer names.

## FILES

**/usr/spool/lp**

## SEE ALSO

**lp**(1), **lpr**(1), **lprm**(1), **lpc**(8), **lpd**(8)

**NAME**

lptest – generate lineprinter ripple pattern

**SYNOPSIS**

**lptest** [ *length* [ *count* ] ]

**DESCRIPTION**

**lptest** writes the traditional "ripple test" pattern on standard output. In 96 lines, this pattern will print all 96 printable ASCII characters in each position. While originally created to test printers, it is quite useful for testing terminals, driving terminal ports for debugging purposes, or any other task where a quick supply of random data is needed.

The *length* argument specifies the output line length if the default length of 79 is inappropriate.

The *count* argument specifies the number of output lines to be generated if the default count of 200 is inappropriate. Note that if *count* is to be specified, *length* must be also be specified.

## NAME

ls – list the contents of a directory

## SYNOPSIS

**ls** [ **–aAcCdfFgilLqrRstu1** ] *filename* ...

## SYSTEM V SYNOPSIS

**/usr/5bin/ls** [ **–abcCdfFgilLmnopqrRstux** ] *filename* ...

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

For each *filename* which is a directory, **ls** lists the contents of the directory; for each *filename* which is a file, **ls** repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents.

In order to determine output formats for the **–C**, **–x**, and **–m** options, **/usr/5bin/ls** uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the **terminfo** database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

### Permissions Field

The mode printed under the **–l** option contains 10 characters interpreted as follows. If the first character is:

- **d** entry is a directory;
- **b** entry is a block-type special file;
- **c** entry is a character-type special file;
- **l** entry is a symbolic link;
- **p** entry is a FIFO (also known as "named pipe") special file;
- **s** entry is an **AF_UNIX** address family socket, or
- **–** entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next refers to permissions to others in the same user-group; and the last refers to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, "execute" permission is interpreted to mean permission to search the directory. The permissions are indicated as follows:

- **r** the file is readable;
- **w** the file is writable;
- **x** the file is executable;
- **–** the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has the set-group-id bit set; likewise the owner-execute permission character is given as **s** if the file has the set-user-id bit set.

The last character of the mode (normally **x** or '–') is **t** if the 1000 bit of the mode is on. See **chmod**(1V) for the meaning of this mode. The indications of set-ID and 1000 bits of the mode are capitalized (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

## OPTIONS

**–a** List all entries; in the absence of this option, entries whose names begin with a '.' are *not* listed (except for the super-user, for whom **ls**, but not **/usr/5bin/ls**, normally prints even files that begin with a '.').

**–A** (**ls** only) Same as **–a**, except that '.' and '..' are not listed.

**–c** Use time of last edit (or last mode change) for sorting or printing.

**–C** Force multi-column output, with entries sorted down the columns; for **ls**, this is the default when output is to a terminal.

**–d** If argument is a directory, list only its name (not its contents); often used with **–l** to get the status of a directory.

**–f** Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **–l**, **–t**, **–s**, and **–r**, and turns on **–a**; the order is the order in which entries appear in the directory.

**–F** Mark directories with a trailing slash ('/'), executable files with a trailing asterisk ('*'), symbolic links with a trailing at-sign ('@'), and AF_UNIX address family sockets with a trailing equals sign ('=').

**–g** For **ls**, show the group ownership of the file in a long output. For **/usr/5bin/ls**, print a long listing, the same as **–l**, except that the owner is not printed.

**–i** For each file, print the i-number in the first column of the report.

**–l** List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. If the file is a special file the size field will instead contain the major and minor device numbers. If the time of last modification is greater than six months ago, it is shown in the format '*month date year*'; files modified within six months show '*month date time*'. If the file is a symbolic link the pathname of the linked-to file is printed preceded by '—>'. **/usr/5bin/ls** will print the group in addition to the owner.

**–L** If argument is a symbolic link, list the file or directory the link references rather than the link itself.

**–q** Display non-graphic characters in filenames as the character **?**; for **ls**, this is the default when output is to a terminal.

**–r** Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

**–R** Recursively list subdirectories encountered.

**–s** Give size of each file, including any indirect blocks used to map the file, in kilobytes (**ls**) or 512-byte blocks (**/usr/5bin/ls**).

**–t** Sort by time modified (latest first) instead of by name.

**–u** Use time of last access instead of last modification for sorting (with the **–t** option) and/or printing (with the **–l** option).

**–1** (**ls** only) Force one entry per line output format; this is the default when output is not to a terminal.

SYSTEM V OPTIONS

**–b** Force printing of non-graphic characters to be in the octal **\ddd** notation.

**–m** Stream output format; the file names are printed as a list separated by commas, with as many entries as possible printed on a line.

**–n** The same as **–l**, except that the owner's UID and group's GID numbers are printed, rather than the associated character strings.

**–o** The same as **–l**, except that the group is not printed.

**–p** Put a slash ('/') after each filename if that file is a directory.

**–x** Multi-column output with entries sorted across rather than down the page.

ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **ls**. On entry to **ls**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character

classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

**FILES**

| | |
|---|---|
| **/etc/passwd** | to get user ID's for '**ls –l**' and '**ls –o**'. |
| **/etc/group** | to get group ID for '**ls –g**' and '**/usr/5bin/ls –l**'. |
| **/usr/share/lib/terminfo/*** | to get terminal information for **/usr/5bin/ls**. |

**SEE ALSO**

**chmod**(1V)

**BUGS**

NEWLINE and TAB are considered printing characters in filenames.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as '**ls –s**' is much different than '**ls –s | lpr**'. On the other hand, not doing this setting would make old shell scripts which used **ls** almost certain losers.

None of the above apply to **/usr/5bin/ls**.

Unprintable characters in file names may confuse the columnar output options.

**NAME**

lsw – list TFS whiteout entries

**SYNOPSIS**

**lsw** *filename* ...

**DESCRIPTION**

For each *filename* that is a directory, **lsw** lists the whiteout entries in that directory. These whiteout entries are created by the translucent file service See **tfs**(4S).

**SEE ALSO**

**unwhiteout**(1), **tfs**(4S), **mount_tfs**(8), **tfsd**(8)

## NAME

m4 – macro language processor

## SYNOPSIS

**m4** [ *filename* ] ...

## SYSTEM V SYNOPSIS

**/usr/5bin/m4** [ **–es** ] [ **–B***int* ] [ **–H***int* ] [ **–S***int* ] [ **–T***int* ] [ **–D***name=val* ] [ **–U***name* ] [ *filename* ]...

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**m4** is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is '–', the standard input is read. The processed text is written on the standard output.

Macro calls have the form:

*name*(*argument1*[, *argument2*, ...,] *argumentn*)

The '(' must immediately follow the name of the macro. If the name of a defined macro is not followed by a '(', it is interpreted as a call of the macro with no arguments. Potential macro names consist of letters, digits, and '_', (underscores) where the first character is not a digit.

Leading unquoted SPACE, TAB, and NEWLINE characters are ignored while collecting arguments. Left and right single quotes (‘ ’) are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, the arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be NULL. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

## SYSTEM V OPTIONS

The options and their effects are as follows:

**–e** Operate interactively. Interrupts are ignored and the output is unbuffered.

**–s** Enable line sync output for the C preprocessor (#line ...)

**–B***int* Change the size of the push-back and argument collection buffers from the default of 4,096.

**–H***int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

**–S***int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

**–T***int* Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any **–D** or **–U** flags:

**–D***name*[=*val*]
Define *filename* to be **val** or to be NULL in *val*'s absence.

**–U***name*
Undefine *name*.

## USAGE

### Built-In Macros

**m4** makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are NULL unless otherwise stated.

**define** The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*'th argument. Argument 0 is the name of the macro; missing arguments are replaced by the NULL string.

**undefine** Remove the definition of the macro named in the argument.

**ifdef** If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is NULL. The word *unix* is predefined.

**changequote** Change quote characters to the first and second arguments. **changequote** without arguments restores the original values (that is, ‘ ’).

**divert** **m4** maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to the (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

**undivert** Display immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

**divnum** Return the value of the current output stream.

**dnl** Read and discard characters up to and including the next NEWLINE.

**ifelse** Has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6, 7 and so on. Otherwise, the value is either the last string not used by the above process, or, if it is not present, NULL.

**incr** Return the value of the argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

**eval** Evaluate the argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation); relationals; parentheses.

**len** Return the number of characters in the argument.

**index** Return the position in the first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.

**substr** Return a substring of the first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

**translit** Transliterate the characters in the first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

**include** Return the contents of the file named in the argument.

**sinclude** Is similar to **include**, except that it says nothing if the file is inaccessible.

**syscmd** Execute the system command given in the first argument. No value is returned.

**maketemp** Fill in a string of XXXXX in the argument with the current process ID.

**errprint** Print the argument on the diagnostic output file.

**dumpdef** Print current names and definitions, for the named items, or for all if no arguments are given.

## SYSTEM V USAGE

In the System V version of **m4**, the following built-in macros have added capabilities.

**Built-In Macros**

**define** '$#' is replaced by the number of arguments; $* is replaced by a list of all the arguments separated by commas; $@ is like '$*', but each argument is quoted (with the current quotes).

**changequote** Change quote symbols to the first and second arguments. The symbols may be up to five characters long.

**eval** Additional operators include bitwise '&', '|', '^' and '~'. Octal, decimal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

The System V version of **m4** makes available the following additional built-in macros.

**defn** Return the quoted definition of the argument(s). It is useful for renaming macros, especially built-ins.

**pushdef** Like **define**, but saves any previous definition.

**popdef** Remove current definition of the argument(s), exposing the previous one, if any.

**shift** Return all but the first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

**changecom** Change left and right comment markers from the default # and NEWLINE. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes NEWLINE. With two arguments, both markers are affected. Comment markers may be up to five characters long.

**decr** Return the value of the argument decremented by 1.

**sysval** Return code from the last call to **syscmd**.

**m4exit** Exit immediately from **m4**. Argument 1, if given, is the exit code; the default is 0.

**m4wrap** Argument 1 will be pushed back at final EOF. For example, '**m4wrap("cleanup()")**'.

**traceon** With no arguments, turn on tracing for all macros (including built-ins). Otherwise, turn on tracing for named macros.

**traceoff** Turn off trace globally and for any macros specified. Macros specifically traced by **traceon** can be untraced only by specific calls to **traceoff**.

**SEE ALSO**

cc(1V)

*m4 — A Macro Processor*, in *Programming Utilities and Libraries*

**NOTES**

While the compiler allows 8-bit strings and comments, 8-bits are not allowed anywhere else. The cc(1V) command does not generate or support 8-bit symbol names because, until ANSI C, non-ASCII support was not expected. The ANSI C specification now suggests that string literals and comments can contain any characters from any character code set.

**NAME**

mach – display the processor type of the current host

**SYNOPSIS**

**mach**

**DESCRIPTION**

The **mach** command displays the processor-type of the current Sun host.

**SEE ALSO**

**arch**(1), **machid**(1)

## NAME

machid, sun, iAPX286, m68k, pdp11, sparc, u3b, u3b2, u3b5, u3b15, vax, i386 – return a true exit status if the processor is of the indicated type

## SYNOPSIS

**sun**

**iAPX286**

**m68k**

**pdp11**

**sparc**

**u3b**

**u3b2**

**u3b5**

**u3b15**

**vax**

**i386**

## DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

| | |
|---|---|
| **sun** | True if you are on a Sun system. |
| **iAPX286** | True if you are on a computer using an iAPX286 processor. |
| **i386** | True if you are on a computer using an iAPX386 processor. |
| **m68k** | True if you are on a computer, such as a Sun-3 or a Sun-3x, using an M68000-family processor. |
| **pdp11** | True if you are on a PDP-11. |
| **sparc** | True if you are on a computer, such as a Sun-4, using a SPARC-family processor. |
| **u3b** | True if you are on a 3B20S computer. |
| **u3b2** | True if you are on a 3B2 computer. |
| **u3b5** | True if you are on a 3B5 computer. |
| **u3b15** | True if you are on a 3B15 computer. |
| **vax** | True if you are on a VAX. |

The commands that do not apply will return a false (non-zero) value. These commands are often used within **make**(1) makefiles and shell procedures to increase portability.

## SEE ALSO

**arch**(1), **mach**(1), **make**(1), **sh**(1), test(1V), **true**(1)

## NAME

mail, Mail – read or send mail messages

## SYNOPSIS

**Mail** [ **–deHinNUv** ] [ **–f** [ *filename* | *+folder* ] ] [ **–T** *file* ] [ **–u** *user* ]

**Mail** [ **–dFinUv** ] [ **–h** *number* ] [ **–r** *address* ] [ **–s** *subject* ] *recipient* ...

**/usr/ucb/mail** ...

## DESCRIPTION

**mail** is a comfortable, flexible, interactive program for composing, sending and receiving electronic messages. While reading messages, **mail** provides you with commands to browse, display, save, delete, and respond to messages. While sending mail, **mail** allows editing and reviewing of messages being composed, and the inclusion of text from files or other messages.

Incoming mail is stored in the *system mailbox* for each user. This is a file named after the user in **/var/spool/mail**. **mail** normally looks in this file for incoming messages, but you can use the MAIL environment variable to have it look in a different file. When you read a message, it is marked to be moved to a secondary file for storage. This secondary file, called the *mbox*, is normally the file **mbox** in your home directory. This file can also be changed by setting the MBOX environment variable. Messages remain in the *mbox* file until deliberately removed.

## OPTIONS

If no *recipient* is specified, **mail** attempts to read messages from the system mailbox.

| | |
|---|---|
| **–d** | Turn on debugging output. (Neither particularly interesting nor recommended.) |
| **–e** | Test for presence of mail. If there is no mail, **mail** prints nothing and exits (with a successful return code). |
| **–F** | Record the message in a file named after the first recipient. Override the **record** variable, if set. |
| **–H** | Print header summary only. |
| **–i** | Ignore interrupts (as with the **ignore** variable). |
| **–n** | Do not initialize from the system default **Mail.rc** file. |
| **–N** | Do not print initial header summary. |
| **–U** | Convert **uucp** style addresses to Internet standards. Overrides the **conv** environment variable. |
| **–v** | Pass the –v flag to **sendmail**(8). |
| **–f** [*filename*] | Read messages from *filename* instead of system mailbox. If no *filename* is specified, the *mbox* is used. |
| **–f** *+folder* | Use the file *folder* in the folder directory (same as the **folder** command). The name of this directory is listed in the **folder** variable. |
| **–h** *number* | The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops. |
| **–r** *address* | Pass *address* to network delivery software. All tilde (~) commands are disabled. |
| **–s** *subject* | Set the **Subject** header field to *subject*. |
| **–T** *file* | Print the contents of the *article-id* fields of all messages that were read or deleted on *file* (for the use of network news programs if available). |
| **–u** *user* | Read *user*'s system mailbox. This is only effective if *user*'s system mailbox is not read protected. |

## USAGE

Refer to *SunOS User's Guide: Getting Started* for tutorial information about **mail**.

### Starting Mail

As it starts, **mail** reads commands from a system-wide file (**/usr/lib/Mail.rc**) to initialize certain variables, then it reads from a private start-up file called the *.mailrc* file (it is normally the file **.mailrc** in your home directory, but can be changed by setting the MAILRC environment variable) for your personal commands and variable settings. Most **mail** commands are legal inside start-up files. The most common uses for this file are to set up initial display options and alias lists. The following commands are *not* legal in the start-up file: **!**, **Copy**, **edit**, **followup**, **Followup**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **replyall**, **replysender**, **shell**, and **visual**. Any errors in the start-up file cause the remaining lines in that file to be ignored.

You can use the **mail** command to send a message directly by including names of recipients as arguments on the command line. When no recipients appear on the **mail** command line, it enters command mode, from which you can read messages sent to you. If you list no recipients and have no messages, **mail** prints the message: '**No mail for** *username*' and exits.

When in command mode (while reading messages), you can send messages using the **mail** command.

### Sending Mail

While you are composing a message to send, **mail** is in *input* mode. If no subject is specified as an argument to the command a prompt for the subject is printed. After entering the subject line, **mail** enters *input* mode to accept the text of your message to send.

As you type in the message, **mail** stores it in a temporary file. To review or modify the message, enter the appropriate *tilde escapes*, listed below, at the beginning of an input line.

To indicate that the message is ready to send, type a dot (or EOF character, normally CTRL-D ) on a line by itself. **mail** submits the message to **sendmail**(8) for routing to each *recipient*.

Recipients can be;

- local usernames
- Internet addresses of the form:

  *name*@*domain*
- **uucp**(1C) addresses of the form:

  [*host*!...*host*!]*host*!*username*
- filenames for which you have write permission
- alias groups

If the name of the *recipient* begins with a pipe symbol (|), the remainder of the name is taken as a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as **lpr**(1) to record outgoing mail on paper. An alias group is the name of a list of recipients that is set by the **alias** command, taken from the host's **/etc/aliases** file, or taken from the Network Information Service (NIS) aliases domain. See **aliases**(5) for more information about mail addresses and aliases.

### Tilde Escapes

The following *tilde escape* commands can be used when composing messages to send. Each must appear at the beginning of an input line. The escape character (~), can be changed by setting a new value for the **escape** variable. The escape character can be entered as text by typing it twice.

~! [*shell-command*]
: Escape to the shell. If present, run *shell-command*.

~. Simulate EOF (terminate message input).

**~:** *mail-command*
**~_** *mail-command*
Perform the indicated **mail** command. Valid only when sending a message while reading mail.

**~?** Print a summary of tilde escapes.

**~A** Insert the autograph string **Sign** into the message.

**~a** Insert the autograph string **sign** into the message.

**~b** *name ...*
Add the *name*s to the blind carbon copy (**Bcc**) list. This is like the carbon copy (**Cc**) list, except that the names in the **Bcc** list are not shown in the header of the mail message.

**~c** *name ...*
Add the *name*s to the carbon copy (**Cc**) list.

**~d** Read in the *dead.letter* file. The name of this file is listed in the variable **DEAD**.

**~e** Invoke the editor to edit the message. The name of the editor is listed in the **EDITOR** variable. The default editor is ex(1).

**~f** [*message-list*]
Forward the listed messages, or the current message being read. Valid only when sending a message while reading mail; the messages are inserted without alteration (as opposed to the **~m** escape).

**~h** Prompt for the message header lines: **Subject**, **To**, **Cc**, and **Bcc**. If the header line contains text, you can edit the text by backspacing over it and retyping.

**~i** *variable*
Insert the value of the named *variable* into the message.

**~m** [*message-list*]
Insert text from the specified messages, or the current message, into the letter. Valid only when sending a message while reading mail; the text the message is shifted to the right, and the string contained in the **indentprefix** variable is inserted as the leftmost characters of each line. If **indentprefix** is not set, a TAB character is inserted into each line.

**~p** Print the message being entered.

**~q** Quit from input mode by simulating an interrupt. If the body of the message is not empty, the partial message is saved in the *dead.letter* file.

**~r** *filename*
**~<** *filename*
**~<!** *shell-command*
Read in text from the specified file or the standard output of the specified *shell-command*.

**~s** *subject*
Set the subject line to *subject*.

**~t** *name ...*
Add each *name* to the list of recipients.

**~v** Invoke a visual editor to edit the message. The name of the editor is listed in the **VISUAL** variable. The default visual editor is vi(1).

**~w** *filename*
Write the message text onto the given file, without the header.

**~x** Exit as with **~q** but do not save the message in the *dead.letter* file.

**~|** *shell-command*
Pipe the body of the message through the given *shell-command*. If *shell-command* returns a successful exit status, the output of the command replaces the message.

**Reading Mail**

When you enter *command* mode in order to read your messages, **mail** displays a header summary of the first several messages, followed by a prompt for one of the commands listed below. The default prompt is the & (ampersand character).

Message are listed and referred to by number. There is, at any time, a **current** message, which is marked by a > in the header summary. For commands that take an optional list of messages, if you omit a message number as an argument, the command applies to the current message.

A *message-list* is a list of message specifications, separated by SPACE characters, which may include:

| | |
|---|---|
| . | The current message. |
| *n* | Message number *n*. |
| ^ | The first undeleted message. |
| $ | The last message. |
| + | The next undeleted message. |
| – | The previous undeleted message. |
| * | All messages. |
| *n–m* | An inclusive range of message numbers. |
| *user* | All messages from *user*. |
| */string* | All messages with *string* in the subject line (case ignored). |
| *:c* | All messages of type *c*, where *c* is one of: |

| | |
|---|---|
| **d** | deleted messages |
| **n** | new messages |
| **o** | old messages |
| **r** | read messages |
| **u** | unread messages |

Note: the context of the command determines whether this type of message specification makes sense.

Additional arguments are treated as strings whose usage depends on the command involved. Filenames, where expected, are expanded using the normal shell filename-substitution mechanism.

Special characters, recognized by certain commands, are documented with those commands.

**Commands**

While in command mode, if you type in an empty command line (a RETURN or NEWLINE only), the print command is assumed. The following is a complete list of **mail** commands:

**!** *shell-command* Escape to the shell. The name of the shell to use is listed in the **SHELL** variable.

**#** *arguments* Null command. This may be used as if it were a comment in *.mailrc* files, but note that it must be separated from its arguments (commentary) by white space.

**=** Print the current message number.

**?** Print a summary of commands.

**alias** [*alias recipient ...*]
**group** [*alias recipient ...*]
Declare an alias for the given list of recipients. The list will be substituted when the *alias* is used as a recipient while sending mail. When put in the *.mailrc* file, this command provides you with a record of the alias. With no arguments, the command displays the list of defined aliases.

**alternates** *name ...* Declare a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **alternates** prints the current list of alternate names.

**cd**[ *directory*]
**chdir** [*directory*] Change directory. If *directory* is not specified, $HOME is used.

**copy** [*message-list*] [*filename*]
Copy messages to the file without marking the messages as saved. Otherwise equivalent to the save command.

**Copy** [*message-list*] Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the **Save** command.

**delete** [*message-list*] Delete messages from the system mailbox. If the variable **autoprint** is set, print the message following the last message deleted.

**discard** [*header-field*...]
**ignore** [*header-field*...]
Suppress printing of the specified header fields when displaying messages on the screen, such as "Status" and "Received". The fields are included when the message is saved unless the variable **alwaysignore** is set. The **Print** and **Type** commands display all header fields, ignored or not.

**dp** [*message-list*]
**dt** [*message-list*]
Delete the specified messages from the system mailbox, and print the message after the last one deleted. Equivalent to a **delete** command followed by a **print** command.

**echo** [*string* ...] Echo the given strings (like **echo**(1V)).

**edit** [*message-list*] Edit the given messages. The messages are placed in a temporary file and the **EDITOR** variable is used to get the name of the editor. The default editor is **ex**(1).

**exit**
**xit** Exit from **mail** without changing the system mailbox. No messages are saved in the *mbox* (see also **quit**).

**file** [*filename*]
**folder** [*filename*] Quit from the current mailbox file and read in the named mailbox file. Several special characters are recognized when used as file names:

| | |
|---|---|
| *%* | Your system mailbox. |
| *%user* | The system mailbox for *user*. |
| *#* | The previous mail file. |
| *&* | Your *mbox* file (of messages previously read). |
| *+filename* | The named file in the *folder* directory (listed in the **folder** variable). |

With no arguments, **file** prints the name of the current mail file, and the number of messages and characters it contains.

**folders** Print the name of each mail file in the *folder* directory (listed in the **folder** variable).

**followup** [*message*] Respond to a message, recording the response in a file, name of which is derived from the author of the message (overrides the **record** variable, if set). See also the **Followup**, **Save**, and **Copy** commands and the **outfolder** variable.

**Followup** [*message-list*]
Respond to the first message in the message list, sending the message to the author of each message in the list. The subject line is taken from the first message, and the response is recorded in a file, the name of which is derived from the author of the first message (overrides the **record** variable, if set). See also the **followup**, **Save**, and **Copy** commands and the **outfolder** variable.

**from** [*message-list*] Print the header summary for the indicated messages or the current message.

**group** *alias name ...* Same as the **alias** command.

**headers** [*message*] Print the page of headers that includes the message specified, or the current message. The **screen** variable sets the number of headers per page. See also the **z** command.

**help** Print a summary of commands.

**hold** [*message-list*]
**preserve** [*message-list*]
Hold the specified messages in the system mailbox.

**if** *s* | *r* | *t*
*mail-command*
...
**else**
*mail-command*
...
**endif** Conditional execution, where *s* will execute following *mail-command* up to an **else** or **endif**, if the program is in *send* mode, *r* executes the *mail-command* only in *receive* mode, and *t* executes the *mail-command* only if **mail** is being run from a terminal. Useful primarily in the *.mailrc* file.

**ignore** [*header-field...*]
Same as the **discard** command.

**inc** Incorporate messages that arrive while you are reading the system mailbox. The new messages are added to the message list in the current **mail** session. This command does not commit changes made during the session, and prior messages are not renumbered.

**list** Prints all commands available. No explanation is given.

**load** [*message*] *filename*
Load the specified message from the name file. *filename* should contain a single mail message including mail headers (as saved by the save command).

**mail** *recipient ...* Mail a message to the specified recipients.

**mbox** [*message-list*] Arrange for the given messages to end up in the standard *mbox* file when **mail** terminates normally. See also the **exit** and **quit** commands.

**new** [*message-list*]
New [*message-list*]
**unread** [*message-list*]
Unread [*message-list*]
Take a message list and mark each message as *not* having been read.

**next** *message* Go to next message matching *message*. A *message-list* can be given instead of *message*, but only first valid message in the list is used. (This can be used, for instance, to jump to the next message from a specific user.)

**pipe** [*message-list*] [*shell-command*]
| [*message-list*] [*shell-command*]
Pipe the message through *shell-command*. The message is treated marked as read (and normally saved to the *mbox* file when **mail** exits). If no arguments are given, the current message is piped through the command specified by the value of the **cmd** variable. If the **page** variable is set, a form feed character is inserted after each message.

**preserve** [*message-list*]
Same as the **hold** command.

**print** [*message-list*]
**type** [*message-list*] Print the specified messages. If the **crt** variable is set, messages longer than the number of lines it indicates paged through the command specified by the **PAGER** variable. The default paging command is **more**(1).

**Print** [*message-list*]
**Type** [*message-list*] Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ignore** and **retain** commands.

**quit** Exit from **mail** storing messages that were read in the *mbox* file and unread messages in the system mailbox. Messages that have been explicitly saved in a file are deleted unless the variable **keepsave** is set.

**reply** [*message-list*]
**respond** [*message-list*]
**replysender** [*message-list*]
Send a response to the author of each message in the *message-list*. The subject line is taken from the first message. If **record** is set to a filename, a copy of the reply is added to that file. If the **replyall** variable is set, the actions of **Reply**/**Respond** and **reply**/**respond** are reversed. The **replysender** command is not affected by the **replyall** variable, but sends each reply only to the sender of each message.

**Reply** [*message*]
**Respond** [*message*]
**replyall** [*message*] Reply to the specified message, including all other recipients of that message. If the variable **record** is set to a filename, a copy of the reply added to that file. If the **replyall** variable is set, the actions of **Reply**/**Respond** and **reply**/**respond** are reversed. The **replyall** command is not affected by the **replyall** variable, but always sends the reply to all recipients of the message.

**retain** Add the list of header fields named to the *retained list*. Only the header fields in the retain list are shown on your terminal when you print a message. All other header fields are suppressed. The set of retained fields specified by the **retain** command overrides any list of ignored fields specified by the **ignore** command. The **Type** and **Print** commands can be used to print a message in its entirety. If **retain** is executed with no arguments, it lists the current set of retained fields.

**save** [*message-list*] [*filename*]
Save the specified messages in the named file. The file is created if it does not exist. If no *filename* is specified, the file named in the **MBOX** variable is used, **mbox** in your home directory by default. Each saved message is deleted from the system mailbox when **mail** terminates unless the **keepsave** variable is set. See also the **exit** and **quit** commands.

**Save** [*message-list*] Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken from the author's name, with all network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and the **outfolder** variables.

**set** [*variable*[=*value*]]
Define a *variable*. To assign a *value* to *variable*, separate the variable name from the value by an '=' (there must be no space before or after the '='). A variable may be given a null, string, or numeric *value*. To embed SPACE characters within a *value* enclose it in quotes.

With no arguments, **set** displays all defined variables and any values they might have. See **Variables** for a description of all predefined **mail** variables.

**shell** Invoke the interactive shell listed in the **SHELL** variable.

**size** [*message-list*] Print the size in characters of the specified messages.

**source** *filename* Read commands from the given file and return to command mode.

**top** [*message-list*] Print the top few lines of the specified messages. If the **toplines** variable is set, it is taken as the number of lines to print. The default number is 5.

**touch** [*message-list*] Touch the specified messages. If any message in *message-list* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See also the **exit** and **quit** commands.

**type** [*message-list*] Same as the **print** command.

**Type** [*message-list*] Same as the **Print** command.

**undelete** [*message-list*]
Restore deleted messages. This command only restores messages *deleted in the current mail session*. If the **autoprint** variable is set, the last message restored is printed.

**unread** [*message-list*]
**Unread** [*message-list*]
Same as the **new** command.

**unset** *variable* ... Erase the specified variables. If the variable was imported from the environment (that is, an environment variable or exported shell variable), it cannot be unset from within **mail**.

**version** Print the current version and release date of the **mail** utility.

**visual** [*message-list*] Edit the given messages with the screen editor listed in the **VISUAL** variable. The default screen editor is **vi**(1). Each message is placed in a temporary file for editing.

**write** [*message-list*] [*filename*]
Write the given messages onto the specified file, but without the header and trailing blank line. Otherwise, this is equivalent to the **save** command.

**xit** Same as the **exit** command.

**z**[+|−] Scroll the header display forward (+) or backward (−) one screenfull. The number of headers displayed is set by the **screen** variable.

**Forwarding Messages**

To forward a specific message, include it in a message to the desired recipients with the **~f** or **~m** tilde escapes. To forward mail automatically, add a comma-separated list of addresses for additional recipients to the **.forward** file in your home directory. This is different from the format of the **alias** command, which takes a space-separated list instead. Note: forwarding addresses must be valid (as described in **aliases**(5)), or the messages will "bounce." You cannot, for instance, reroute your mail to a new host by forwarding it to your new address if it is not yet listed in the NIS aliases domain.

**Variables**

The behavior of **mail** is governed by a set of predefined variables that are set and cleared using the **set** and **unset** commands.

*Environment Variables*

Values for the following variables are read in automatically from the environment; they cannot be altered from within **mail**:

**HOME**=*directory*
The user's home directory.

**MAIL**=*filename*
The name of the initial mailbox file to read (in lieu of the standard system mailbox). The default is **/var/spool/mail/***username* .

**MAILRC**=*filename*

The name of the personal start-up file. The default is **$HOME/.mailrc**.

*Mail Variables*

The following variables can be initialized within the *.mailrc* file, or set and altered interactively using the **set** command. They can also be imported from the environment (in which case their values cannot be changed within **mail**). The **unset** command clears variables. The set command can also be used to clear a variable by prefixing the word **no** to the name of the variable to clear.

Variables for which values are normally supplied are indicated with an equal-sign (=). The equal-sign is required by the **set** command, and there can be no spaces between the variable-name, equal-sign, and value, using **set** to assign a value.

**allnet** All network names whose last component (login name) match are treated as identical. This causes the message list specifications to behave similarly. Default is **noallnet**. See also the **alternates** command and the **metoo** variable.

**alwaysignore** Ignore header fields with **ignore** everywhere, not just during **print** or **type**. Affects the **save**, **Save**, **copy**, **Copy**, **top**, **pipe**, and **write** commands, and the **~m** and **~f** tilde escapes.

**append** Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend** but **append** is set in the global start-up file (which can be suppressed with the **-n** command line option).

**askcc** Prompt for the **Cc** list after message is entered. Default is **noaskcc**.

**asksub** Prompt for subject if it is not specified on the command line with the **-s** option. Enabled by default.

**autoprint** Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint**.

**bang** Enable the special-casing of exclamation points (!) in shell escape command lines as in **vi**(1). Default is **nobang**.

**cmd**=*shell-command*

Set the default command for the **pipe** command. No default value.

**conv**=*conversion*

Convert **uucp** addresses to the address style specified by *conversion*, which can be either:

**internet**

This requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing.

**optimize**

Remove loops in **uucp**(1C) address paths (typically generated by the **reply** command). No rerouting is performed; **mail** has no knowledge of UUCP routes or connections.

Conversion is disabled by default. See also **sendmail**(8) and the **-U** command line option.

**crt**=*number* Pipe messages having more than *number* lines through the command specified by the value of the **PAGER** variable (*more* by default). Disabled by default.

**DEAD**=*filename*

The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is the file **dead.letter** in your home directory.

**debug** Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

**dot** Take a period on a line by itself during input from a terminal as EOF. Default is **nodot** but **dot** is set in the global start-up file (which can be suppressed with the **–n** command line option).

**editheaders** Include message headers in the text to be edited by the ~**e** and ~**v** commands.

**EDITOR**=*shell-command*
The command to run when the edit or ~**e** command is used. Default is **ex**(1).

**escape**=*c* Substitute *c* for the ~ escape character.

**folder**=*directory*
The directory for saving standard mail files. User specified file names beginning with a plus (+) are expanded by preceding the filename with this directory name to obtain the real filename. If *directory* does not start with a slash (/), the value of **HOME** is prepended to it. There is no default for the **folder** variable. See also **outfolder** below.

**header** Enable printing of the header summary when entering **mail**. Enabled by default.

**hold** Preserve all messages that are read in the system mailbox instead of putting them in the standard *mbox* save file. Default is **nohold** for **mail** and **hold** for **mailtool**(1).

**ignore** Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

**ignoreeof** Ignore EOF during message input. Input must be terminated by a period ('.') on a line by itself or by the '~.' command. Default is **noignoreeof**. See also **dot** above.

**indentprefix**=*string*
When **indentprefix** is set, *string* is used to mark indented lines from messages included with ~**m**. The default is a TAB character.

**keep** When the system mailbox is empty, truncate it to zero length instead of removing it. Disabled by default.

**keepsave** Keep messages that have been saved in other files in the system mailbox instead of deleting them. Default is **nokeepsave**.

**LISTER**=*shell-command*
The command (and options) to use when listing the files in the **folder** directory. The default is ls(1V).

**MBOX**=*filename*
The name of the file to save messages which have been read. The **xit** command overrides this variable, as does saving the message explicitly to another file. Default is the file **mbox** in your home directory.

**metoo** If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

**no** When used as a prefix to a variable name, has the effect of unsetting the variable.

**onehop** When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (that is, one "hop" away).

**outfolder** Locate the files used to record outgoing messages in the directory specified by the **folder** variable unless the pathname is absolute. Default is **nooutfolder**. See **folder** above and the **Save**, **Copy**, **followup**, and **Followup** commands.

**page** Used with the **pipe** command to insert a form feed after each message sent through the pipe. Default is **nopage**.

**PAGER**=*shell-command*
The command to use as a filter for paginating output, along with any options to be used. Default is **more**(1).

**prompt**=*string* Set the *command mode* prompt to *string*. Default is '**&**'.

**quiet** Refrain from printing the opening message and version when entering **mail**. Default is **noquiet**.

**record**=*filename*
Record all outgoing mail in *filename*. Disabled by default. See also the variable **outfolder**.

**replyall** Reverse the effect of the **reply** and **Reply** commands.

**save** Enable saving of messages in the *dead.letter* file on interrupt or delivery error. See **DEAD** for a description of this file. Enabled by default.

**screen**=*number* Set the number of lines in a screen–full of headers for the **headers** command.

**sendmail**=*shell-command*
Alternate command for delivering messages. Note: in addition to the expected list of recipients, **mail** also passes the **–i** and **–m**, flags to the command. Since these flags are not appropriate to other commands, you may have to use a shell script that strips them from the arguments list before invoking the desired command.

**sendwait** Wait for background mailer to finish before returning. Default is **nosendwait**.

**SHELL**=*shell-command*
The name of a preferred command interpreter. Typically inherited from the environment, the shell is normally the one you always use. Otherwise defaults to **sh**(1).

**showto** When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

**sign**=*autograph* The *autograph* text inserted into the message when the **~a** (autograph) command is given. No default (see also the **~i** tilde escape).

**Sign**=*autograph* The *autograph* text inserted into the message when the **~A** command is given. No default (see also the **~i** tilde escape).

**toplines**=*number*
The number of lines of header to print with the **top** command. Default is 5.

**verbose** Invoke **sendmail** with the **–v** flag.

**VISUAL**=*shell-command*
The name of a preferred screen editor. Default is **vi**.

**FILES**

| | |
|---|---|
| **$HOME/***.mailrc* | personal start-up file |
| **$HOME/***.forward* | list of recipients for automatic forwarding of messages |
| **$HOME/***mbox* | secondary storage file |
| **$HOME/***dead.letter* | undeliverable messages file |
| **/var/spool/mail** | directory for system mailboxes |
| **/usr/lib/Mail.help*** | help message files |
| **/usr/lib/Mail.rc** | global start-up file |
| **/tmp/R[emqsx]*** | temporary files |

SEE ALSO

**biff**(1), **bin-mail**(1), **echo**(1V), **ex**(1), **fmt**(1), **ls**(1V), **mailtool**(1), **more**(1), **sh**(1), **uucp**(1C), **vacation**(1), **vi**(1), **aliases**(5), **newaliases**(8), **sendmail**(8)

*SunOS User's Guide: Getting Started*

**mail** is found in **/usr/ucb/Mail**, as a link to **/usr/ucb/mail**. If you wish to use the original (version 7) UNIX mail program, you can find it in **/usr/bin/mail**. Its man page is named **bin-mail**(1).

BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

Replies do not always generate correct return addresses. Try resending the errant reply with **onehop** set.

**mail** does not lock your record file. So, if you use a record file and send two or more messages simultaneously, lines from the messages may be interleaved in the record file.

The format for the **alias** command is a space-separated list of recipients, while the format for an alias in either the **.forward** or **/etc/aliases** is a comma-separated list.

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

## NAME

mailtool – SunView interface for the mail program

## SYNOPSIS

**mailtool** [ **–Mx** ] [ **–Mi** *interval* ] [ *generic-tool-arguments* ]

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**mailtool** is the standard SunView interface to **mail**(1). It provides a window and mouse-based interface for reading, storing, composing, and sending mail messages. Scrollable windows allow easy access to your mailbox and mail folders. Software "panel buttons" make frequently-used commands readily available. Less-used commands are accessible from menus, and keyboard accelerators are provided for the more experienced user.

The full editing capabilities of **textedit**(1) and the SunView selection service are available for modifying and composing mail. In addition, you can customize **mailtool** by setting various parameters with **defaultsedit**(1).

## OPTIONS

**–Mx** Expert mode. Do not ask for confirmation after potentially damaging **mail** commands. This has the same effect as setting the **expert** variable.

**–Mi** *interval* Check for new mail every *interval* seconds. This has the same effect as specifying a value for the *interval* variable.

*generic-tool-arguments*

**mailtool** accepts the generic tool arguments described in **sunview**(1).

## USAGE

Users who are not familiar with the **mail** command should read *SunOS User's Guide: Getting Started*. For more information on text editing and the selection service, see the *SunView User's Guide*.

**mailtool** comes up closed. You can open the tool by clicking on the icon with the LEFT mouse button. **mailtool** starts reading your system mailbox as it opens. Alternatively, the frame menu on the icon contains the **Open** pull-right item, which allows you to open **mailtool** to a selected folder, or open and **Compose**, or just open without performing any other operation.

### Subwindows

**mailtool** is composed of six parts. From top to bottom, they are:

frame header

This is the broad stripe at the top of the tool, and it displays status information. The right side displays information about the most-recently executed command. The left side displays other information such as the name of the current folder. When involved in a lengthy operation, **mailtool** displays a message to that effect on the left side. While an operation is pending, the cursor takes on the shape of an hourglass; you must wait until it finishes.

header-list window

This read-only text window contains a list of message headers from the current folder or mailbox. Initially, it shows the contents of your system mailbox (by default). Each header typically contains fields indicating who the message is from, its subject, and so forth. There is a scrollbar to the left that you can use to scroll through the headers.

control panel

This panel contains a collection of software buttons corresponding to the most frequently used **mail**(1) commands. Clicking on one of these buttons starts the indicated operation for either the selected or the current message. Commands that require the name of a folder, such as **Save**, use the contents of the '**File:**' text item. You must enter the name of a file or folder in this space before clicking on the button.

In addition to the panel buttons, other commands and variations are accessible through menus "behind" each panel button. To display a button menu, hold down the RIGHT mouse button over the panel button. See **Command Menus**, below, for details.

The message window

This text window displays the current message (marked with '>' in the header window). You can edit this message, in which case the result replaces the original message in the mailbox or folder.

composition panel

This panel contains software buttons for composing or replying to messages, and is visible only when you are composing a message or reply.

composition window

This text window, in which you compose messages to send, appears in conjunction with the composition panel. It is normally displayed only when composing a message or reply. It is initially loaded with mail header lines such as '**To:**' and '**Subject:**', and perhaps, '**Cc:**'. After these labels come various text fields, such as **|>recipients<|**. You can move from field to field using CTRL-TAB. This advances to the next text field. If you supply input to the field, your input replaces its contents. Empty fields are deleted when the message is sent. You can continue to edit the message as you see fit until you click on **Deliver**, at which point the message is sent as is. (You normally cannot retrieve a message once it has been sent).

pop-up composition window

While composing a message or reply, clicking again on **Compose** or **Reply**, opens another frame that contains a composition panel and window. The only limit on the number of such pop-up composition frames is the number of windows that a tool can support. These additional frames operate independently.

**Basic Mailtool Concepts**

Choosing a message

To choose a message, place the cursor anywhere in its header in the header window and click the LEFT mouse button. If there is no message chosen, **mailtool** applies operations to the current message.

Current message

The message that is displayed in the message window, and flagged with a '>' in the header window.

Confirmation

Some operations require confirmation, in which case an alert is displayed. You can then confirm or cancel the operation.

Folders

When mailtool starts up, it normally reads your system mailbox. However, you can select another "folder" (file containing mail messages) from which to read.

Committing Changes

Some operations change the state of your system mailbox or the current folder. These changes are not finalized until you commit them. For instance, you can "undelete" messages that were deleted, provided that you have not yet done an operation that commits your changes. If the **mailtool** session is interrupted, pending changes to the mailbox or folder do

not take effect. The **Done** button commits changes, as does the **Quit** button, which also exits from **mailtool**. To deliberately exit without committing, use the pop-up menu behind the **Quit** button.

**Control Panel**

Except for the **Next** and **Undelete** buttons, **mailtool** commands operate on the selected message or the current message only. You cannot specify a list or range of messages as with **mail**(1). The control panel buttons and items are (in alphabetical order):

**Compose** Open a composition panel and window to compose a message.

**Delete** Delete the selected or current message.

**Done** Commit changes, close **mailtool**, and read new mail on next **Open**.

**File:** This is a text item in which to enter the name of a folder for the **Save**, **Copy**, and **Folder** commands. This name can be a full pathname, a pathname relative to the current directory (the directory **mailtool** was started from), or a filename prefixed with a '+' to refer to a file in the "folder" directory.

**Folder** Commit changes and switch to the file or folder specified in the '**File:**' text item.

**Misc** Display a pop-up panel to change the current directory of **mailtool**. Other miscellaneous operations are available on the menu behind this button.

**New Mail** If you are examining your system mailbox, retrieve new mail *without* committing changes. If your are examining a folder, commit any changes to the folder, switch back to system mailbox, and retrieve any new mail in the process.

**Next** Display the message following the *current* message in the message window.

**Print** Print the corresponding message on a hardcopy printer.

**Reply** Open a composition window to reply to the selected or current message.

**Save** Save the current or selected message in the folder specified in the '**File:**' text field, and delete it from your system mailbox or current folder.

**Show** Display the chosen or current message in the message window.

**The Composition Panel**

This panel contains four buttons and a cycle-item. The cycle-item controls the behavior of the composition window when it becomes inactive — when the user delivers or cancels a message. Items in the cycle are:

**Disappear** Remove the composition window and panel from display. This is the default.

**Stay Up** Clear the window, but leave it displayed.

**Close** Close a pop-up composition frame.

The panel buttons are:

**Cancel** Abort the message being composed.

**Deliver** Send the message being composed to the indicated recipients.

**Include** Insert the corresponding message into the composition window at the caret. This operation can be performed repeatedly, to include various messages.

**Re-address** Insert the appropriate '**To:**', '**Subject:**' and '**Cc:**' fields at the top of the composition window.

**Command Menus**

All panel buttons have menus behind them. The first item on the menu is the default command; choosing this item is the same as clicking on the panel button.

Some menu items are pull-right to menus of related commands. You can browse the button menus to discover what additional commands are available, and what their accelerators are, if any. The following commands are particularly useful.

**Change Directory**
Display a pop-up panel to change the current directory.

**Commit Changes**
Commit changes. This item is behind the **New Mail** button when viewing a folder.

**Commit Changes and Quit**
Behind the **Done** button. Commit changes and exit **mailtool**(1). This is the same as choosing **Quit** from the frame menu.

**Commit Changes and Retrieve New Mail**
Behind the **New Mail** button. Commit changes and retrieve new mail, switching to the system mail box if in a folder. This is the default when viewing a folder.

**Copy** Behind the **Save** button. Copy the selected message to the file or folder specified in the '**File:**' text item, without deleting it from the mailbox or folder.

**Deliver, Leave Window Intact**
Behind the **Deliver** button. Deliver the message, but do not undisplay, close, or clear the message composition window.

**Include, Indented**
Behind the **Include** button. Include the indicated message, setting it off by indentation rather than bracketing it with '**--- Begin Included Message ---**' and '**--- End Included Message ---**' lines.

**Previous** Behind the **Next** button. Display the previous message in the message window.

**Quit without Committing Changes**
Behind the **Done** button. Exit **mailtool** *without* committing changes.

**Show Full Header**
Behind the **Show** button. Display the complete message in the message window, including header lines that are normally ignored.

**Source .mailrc** Behind the **Misc** button. Read in your **.mailrc** file to acquire new variables and settings. Note: this operation does not "forget" the previous option settings; only changes to boolean variables take effect.

**Undelete** Behind the **Delete** button. Undelete the most recently deleted message(s) — this may be used repeatedly. It is inactive when there are no deleted messages.

There are two special menus for use with the '**File:**' text item. Choosing a name from either of these menus replaces the contents of this item. The menu behind the '**File:**' item holds the most recently used folder names of the current session. It is be initialized by the **filemenu** variable. The menu behind the **Folder** button displays all folders in the "folder" directory, which is specified by the **folder** variable (described in **mail**(1)). Folders can be organized into subdirectories within the folder directory. Files in these subdirectories appear in a hierarchy of pull-right menus.

To switch to a folder, choose it from one of the file menus, or type it in directly, and click on the **Folder** button. To return to your system mailbox, use the **New Mail** button.

### Mailtool Variables

In addition to the variables recognized by **mail**(1), **mailtool** recognizes those listed below. They can be set by using **defaultsedit**(1), or by editing your **.mailrc** file directly. Unless otherwise noted, the default for the following variables is off.

**allowreversescan**
When set, allows you to step through messages in latest-first oldest-first order if you choose. The next message depends on the order of travel.

**alwaysusepopup** Never split the message window to compose or reply; always use pop-up composition windows.

**askbcc** Prompt for the '**Bcc:**' field when composing or replying.

**autoprint** Display the next message when the current message is deleted or saved.

**bell** The number of times to ring the bell when new mail arrives. The default is 0.

**disablefields** Do not use text fields in the composition window. The default is to use text fields.

**editmessagewindow**
Request confirmation before the first editing operation to a message in the message window (as opposed to composing a reply). The default is not to request confirmation of the first edit.

**expert** Set expert mode in which no confirmations are requested.

**filemenu** A list of files from which to initialize the '**File:**' menu. These can be absolute pathnames, pathnames relative to the working directory for **mailtool** (typically your home directory), or filenames prefixed with a '+', which are taken as relative to the directory specified in the **folder** variable (see **mail**(1)).

**filemenusize** Specifies the maximum size of the '**File:**' menu. The default is 10.

**flash** The number of times to flash the window or icon when new mail arrives. The default is 0.

**headerlines** The number of lines in header window. The default is 10.

**interval** The interval in seconds to check for new mail. The default is 300.

**maillines** The number of lines in mail message window. The default is 30.

**moveinputfocus** Move the input focus into the composition window for **Compose** and **Reply**. This only works for click-to-type.

**pop-uplines** The number of lines in pop-up message composition window. The default is 30.

**msgpercent** The percent of the message window to remain visible during **Compose** or **Reply**. The default is 50.

**printmail** The command to use to print a message. The default is '**lpr -p**'.

**trash** The name of trash bin, which may be accessed just like any other folder. If set, all deleted messages are moved to the trash bin. The trash bin is emptied when you commit changes.

### Conditional Settings

You can make your **.mailrc** set variables conditionally, depending on whether it is running in the tty environment or the window environment. See *SunOS User's Guide: Getting Started* for details.

## ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **mailtool**. On entry to **mailtool**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default.** When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

**FILES**

| | |
|---|---|
| **/var/spool/mail/*** | system mailboxes |
| **~/.mailrc** | startup file for **mail** and **mailtool** |

**SEE ALSO**

**bin-mail**(1), **defaultsedit**(1), **mail**(1), **sunview**(1), **textedit**(1) **aliases**(5), **locale**(5), **newaliases**(8), **sendmail**(8)

*SunOS User's Guide: Getting Started*
*SunView User's Guide*

**BUGS**

If **mail**(1) receives an error, then **mailtool** may hang, in which case you must kill it.

New mail status is only approximate, therefore the presence of new mail is not always accurately reflected in the icon image or tool frame header.

Mouse input may be lost while **mailtool** switches to iconic state.

When notifying you of new mail, **mailtool** will not flash the window or icon without beeping (ringing the audible bell). Thus, the number of flashes is limited by the number of beeps you set.

Unlike **mail**(1), **mailtool** retains unsaved messages in the system mailbox by default; that is, the **hold** variable is initially set.

## NAME

make – maintain, update, and regenerate related programs and files

## SYNOPSIS

**make** [ **–f** *makefile* ] ... [ **–d** ] [ **–dd** ] [ **–D** ] [ **–DD** ] [ **–e** ] [ **–i** ] [ **–k** ] [ **–n** ] [ **–p** ] [ **–P** ] [ **–q** ] [ **–r** ] [ **–s** ] [ **–S** ] [ **–t** ] [ *target* ... ] [ *macro=value* ... ]

## DESCRIPTION

**make** executes a list of shell commands associated with each *target*, typically to create or update a file of the same name. *makefile* contains entries that describe how to bring a target up to date with respect to those on which it depends, which are called *dependencies*. Since each dependency is a target, it may have dependencies of its own. Targets, dependencies, and sub-dependencies comprise a tree structure that **make** traces when deciding whether or not to rebuild a *target*.

**make** recursively checks each *target* against its dependencies, beginning with the first target entry in *makefile* if no *target* argument is supplied on the command line. If, after processing all of its dependencies, a target file is found either to be missing, or to be older than any of its dependencies, **make** rebuilds it. Optionally with this version of **make**, a target can be treated as out-of-date when the commands used to generate it have changed since the last time the target was built.

To build a given target, **make** executes the list of commands, called a *rule*. This rule may be listed explicitly in the target's makefile entry, or it may be supplied implicitly by **make**.

When no *makefile* is specified with a **–f** option:

- If there is a file named **makefile** in the working directory, **make** uses that file. If, however, there is an SCCS history file (**SCCS/s.makefile**) which is newer, **make** attempts to retrieve and use the most recent version.
- In the absence of the above file(s), if a file named **Makefile** is present in the working directory, **make** attempts to use it. If there is an SCCS history file (**SCCS/s.Makefile**) that is newer, **make** attempts to retrieve and use the most recent version.

If no *target* is specified on the command line, **make** uses the first target defined in *makefile*.

If a *target* has no makefile entry, or if its entry has no rule, **make** attempts to derive a rule by each of the following methods, in turn, until a suitable rule is found. (Each method is described under USAGE below.)

- Pattern matching rules.
- Implicit rules, read in from a user-supplied makefile.
- Standard implicit rules (also known as suffix rules), typically read in from the file **<make/default.mk>**.
- SCCS retrieval. **make** retrieves the most recent version from the SCCS history file (if any). See the description of the **.SCCS_GET:** special-function target for details.
- The rule from the **.DEFAULT:** target entry, if there is such an entry in the makefile.

If there is no makefile entry for a *target*, if no rule can be derived for building it, and if no file by that name is present, **make** issues an error message and halts.

## OPTIONS

**–f** *makefile*

Use the description file *makefile*. A '–' as the *makefile* argument denotes the standard input. The contents of *makefile*, when present, override the standard set of implicit rules and predefined macros. When more than one '**–f** *makefile*' argument pair appears, **make** uses the concatenation of those files, in order of appearance.

**–d** Display the reasons why **make** chooses to rebuild a target; **make** displays any and all dependencies that are newer. In addition, **make** displays options read in from the **MAKEFLAGS** environment variable.

**–dd** Display the dependency check and processing in vast detail.

**–D** Display the text of the makefiles read in.

**–DD** Display the text of the makefiles, **default.mk** file, the state file, and all hidden-dependency reports.

**–e** Environment variables override assignments within makefiles.

**–i** Ignore error codes returned by commands. Equivalent to the special-function target '**.IGNORE:**'.

**–k** When a nonzero error status is returned by a rule, or when **make** cannot find a rule, abandon work on the current target, but continue with other dependency branches that do not depend on it.

**–n** No execution mode. Print commands, but do not execute them. Even lines beginning with an @ are printed. However, if a command line contains a reference to the **$(MAKE)** macro, that line is always executed (see the discussion of **MAKEFLAGS** in **Reading Makefiles and the Environment**).

**–p** Print out the complete set of macro definitions and target descriptions.

**–P** Merely report dependencies, rather than building them.

**–q** Question mode. **make** returns a zero or nonzero status code depending on whether or not the target file is up to date.

**–r** Do not read in the default makefile **<make/default.mk>**.

**–s** Silent mode. Do not print command lines before executing them. Equivalent to the special-function target '**.SILENT:**'.

**–S** Undo the effect of the **–k** option. Stop processing when a non-zero exit status is returned by a command.

**–t** Touch the target files (bringing them up to date) rather than performing their rules. *This can be dangerous when files are maintained by more than one person.* When the **.KEEP_STATE:** target appears in the makefile, this option updates the state file just as if the rules had been performed.

*macro=value*

Macro definition. This definition overrides any regular definition for the specified macro within the makefile itself, or in the environment. However, this definition can still be overridden by conditional macro assignments.

## USAGE

Refer to *SunOS User's Guide: Doing More* and **make** in *Programming Utilities and Libraries* for tutorial information about **make.**

### Reading Makefiles and the Environment

When **make** first starts, it reads the **MAKEFLAGS** environment variable to obtain any the following options specified present in its value: **–d**, **–D**, **–e**, **–i**, **–k**, **–l**, **–n**, **–p**, **–q**, **–r**, **–s**, **–S**, or **–t**. (Within the **MAKEFLAGS** value, the leading '**—**' character for the option string is omitted.) **make** then reads the command line for additional options, which also take effect.

Next, **make** reads in a default makefile that typically contains predefined macro definitions, target entries for implicit rules, and additional rules, such as the rule for retrieving SCCS files. If present, **make** uses the file **default.mk** in the current directory; otherwise it reads the file **<make/default.mk>**, which contains the standard definitions and rules.
Use the directive:

**include <make/default.mk>**

in your local **default.mk** file to include them.

Next, **make** imports variables from the environment (unless the **–e** option is in effect), and treats them as defined macros. Because **make** uses the most recent definition it encounters, a macro definition in the makefile normally overrides an environment variable of the same name. When **–e** is in effect, however, environment variables are read in *after* all makefiles have been read. In that case, the environment variables take precedence over definitions in the makefile.

Next, **make** reads the state file, **.make.state** in the local directory if it exists, and then any makefiles you specify with **–f**, or one of **makefile** or **Makefile** as described above.

Next, (after reading the environment if **–e** is in effect), **make** reads in any macro definitions supplied as command line arguments. These override macro definitions in the makefile and the environment both, but only for the **make** command itself.

**make** exports environment variables, using the most recently defined value. Macro definitions supplied on the command line are not normally exported, unless the macro is also an environment variable.

**make** does not export macros defined in the makefile. If an environment variable is set, and a macro with the same name is defined on the command line, **make** exports its value as defined on the command line. Unless **–e** is in effect, macro definitions within the makefile take precedence over those imported from the environment.

The macros **MAKEFLAGS**, **MAKE**, **KEEP_STATE**, **SHELL**, **HOST_ARCH**, **TARGET_ARCH**, **HOST_MACH**, and **TARGET_MACH** are special cases. See **Special-Purpose Macros** below, for details.

**Makefile Target Entries**

A target entry has the following format:

*target*... [: | ::] [*dependency*] ... [; *command*] ...
[*command*]
...

The first line contains the name of a target, or a space-separated list of target names, terminated with a colon or double colon. If a list of targets is given, this is equivalent to having a separate entry of the same form for each target. The colon(s) may be followed by a *dependency*, or a dependency list. **make** checks this list before building the target. The dependency list may be terminated with a semicolon (;), which in turn can be followed by a single Bourne shell command. Subsequent lines in the target entry begin with a TAB, and contain Bourne shell commands. These commands comprise the rule for building the target.

Shell commands may be continued across input lines by escaping the NEWLINE with a backslash (\). The continuing line must also start with a TAB.

To rebuild a target, **make** expands macros, strips off initial TAB characters and either executes the command directly (if it contains no shell metacharacters), or passes each command line to a Bourne shell for execution.

The first line that does not begin with a TAB or # begins another target or macro definition.

**Makefile Special Characters**

*Global*

**#** Start a comment. The comment ends at the next NEWLINE. If the # follows the TAB in a command line, that line is passed to the shell (which also treats # as the start of a comment).

**include** *filename*

If the word **include** appears as the first seven letters of a line and is followed by a SPACE or TAB, the string that follows is taken as a filename to interpolate at that line. **include** files can be nested to a depth of no more than about 16. If *filename* is a macro reference, it is expanded.

*Targets and Dependencies*

**:** Target list terminator. Words following the colon are added to the dependency list for the target or targets. If a target is named in more than one colon-terminated target entry, the dependencies for all its entries are added to form that target's complete dependency list.

**::** Target terminator for alternate dependencies. When used in place of a ':' the double-colon allows a target to be checked and updated with respect to alternate dependency lists. When the target is out-of-date with respect to dependencies listed in the first alternate, it is built according to the rule for that entry. When out-of-date with respect to dependencies in another alternate, it is built according the rule in that other entry. Implicit rules do not apply to double-colon targets; you must supply a rule for each entry. If no dependencies are specified, the rule is always performed.

*target* [+ *target*...] **:**

Target group. The rule in the target entry builds all the indicated targets as a group. It is normally performed only once per **make** run, but is checked for command dependencies every time a target in the group is encountered in the dependency scan.

**%** Pattern matching wild card metacharacter. Like the '*' shell wild card, '%' matches any string of zero or more characters in a target name or dependency, in the target portion of a conditional macro definition, or within a pattern replacement macro reference. Note: only one '%' can appear in a target, dependency-name, or pattern-replacement macro reference.

*./pathname*

**make** ignores the leading './' characters from targets with names given as pathnames relative to "dot," the working directory.

*Macros*

**=** Macro definition. The word to the left of this character is the macro name; words to the right comprise its value. Leading and trailing white space characters are stripped from the value. A word break following the = is implied.

**$** Macro reference. The following character, or the parenthesized or bracketed string, is interpreted as a macro reference: **make** expands the reference (including the $) by replacing it with the macro's value.

**( )**
**{ }** Macro-reference name delimiters. A parenthesized or bracketed word appended to a $ is taken as the name of the macro being referred to. Without the delimiters, **make** recognizes only the first character as the macro name.

**$$** A reference to the dollar-sign macro, the value of which is the character '$'. Used to pass variable expressions beginning with $ to the shell, to refer to environment variables which are expanded by the shell, or to delay processing of dynamic macros within the dependency list of a target, until that target is actually processed.

**\$** Escaped dollar-sign character. Interpreted as a literal dollar sign within a rule.

**+=** When used in place of '=', appends a string to a macro definition (must be surrounded by white space, unlike '=').

**:=** Conditional macro assignment. When preceded by a list of targets with explicit target entries, the macro definition that follows takes effect when processing only those targets, and their dependencies.

**:sh =** Define the value of a macro to be the output of a command (see **Command Substitutions**, below).

**:sh** In a macro reference, execute the command stored in the macro, and replace the reference with the output of that command (see **Command Substitutions**).

*Rules*

– **make** ignores any nonzero error code returned by a command line for which the first non-TAB character is a '–'. This character is not passed to the shell as part of the command line. **make** normally terminates when a command returns nonzero status, unless the –**i** or –**k** options, or the **.IGNORE:** special-function target is in effect.

**@** If the first non-TAB character is a **@**, **make** does not print the command line before executing it. This character is not passed to the shell.

**?** Escape command-dependency checking. Command lines starting with this character are not subject to command dependency checking.

**!** Force command-dependency checking. Command-dependency checking is applied to command lines for which it would otherwise be suppressed. This checking is normally suppressed for lines that contain references to the '**?**' dynamic macro (for example, '**$?**').

When any combination of '–', '**@**', '**?**', or '**!**' appear as the first characters after the TAB, all that are present apply. None are passed to the shell.

**Special-Function Targets**

When incorporated in a makefile, the following target names perform special-functions:

**.DEFAULT:**
If it has an entry in the makefile, the rule for this target is used to process a target when there is no other entry for it, no rule for building it, and no SCCS history file from which to retrieve a current version. **make** ignores any dependencies for this target.

**.DONE:** If defined in the makefile, **make** processes this target and its dependencies after all other targets are built. This target is also performed when **make** halts with an error, unless the **.FAILED** target is defined.

**.FAILED:**
This target, along with its dependencies, is performed instead of **.DONE** when defined in the makefile and **make** halts with an error.

**.IGNORE:**
Ignore errors. When this target appears in the makefile, **make** ignores non-zero error codes returned from commands.

**.INIT:** If defined in the makefile, this target and its dependencies are built before any other targets are processed.

**.KEEP_STATE:**
If this target appears in the makefile, **make** updates the state file, **.make.state**, in the current directory. This target also activates command dependencies, and hidden dependency checks.

**.MAKE_VERSION:**
A target-entry of the form:

**.MAKE_VERSION: VERSION–***number*

enables version checking. If the version of **make** differs from the version indicated, **make** issues a warning message.

**.NO_PARALLEL:**
Currently, this target has no effect, it is, however, reserved for future use.

**.PARALLEL:**
Currently of no effect, but reserved for future use.

**.PRECIOUS:**
List of files not to delete. **make** does not remove any of the files listed as dependencies for this target when interrupted. **make** normally removes the current target when it receives an interrupt.

**.SCCS_GET:**
This target contains the rule for retrieving the current version of an SCCS file from its history file. To suppress automatic retrieval, add an entry for this target with an empty rule to your makefile.

**.SILENT:**
Run silently. When this target appears in the makefile, **make** does not echo commands before executing them.

**.SUFFIXES:**
The suffixes list for selecting implicit rules (see **The Suffixes List**).

**.WAIT:** Currently of no effect, but reserved for future use.

*Clearing Special Targets*

In this version of **make**, you can clear the definition of the following special targets by supplying entries for them with no dependencies and no rule:

**.DEFAULT**, **.SCCS_GET**, and **.SUFFIXES**

**Command Dependencies**

When the **.KEEP_STATE:** target appears in the makefile, **make** checks the command for building a target against the state file, **.make.state**. If the command has changed since the last **make** run, **make** rebuilds the target.

**Hidden Dependencies**

When the **.KEEP_STATE:** target appears in the makefile, **make** reads reports from **cpp**(1) and other compilation processors for any "hidden" files, such as **#include** files. If the target is out of date with respect to any of these files, **make** rebuilds it.

**Macros**

Entries of the form

*macro=value*

define macros. *macro* is the name of the macro, and *value*, which consists of all characters up to a comment character or unescaped NEWLINE, is the value. **make** strips both leading and trailing white space in accepting the value.

Subsequent references to the macro, of the forms: $(*name*) or ${*name*} are replaced by *value*. The parentheses or brackets can be omitted in a reference to a macro with a single-character name.

Macro references can contain references to other macros, in which case nested references are expanded first.

*Suffix Replacement Macro References*

Substitutions within macros can be made as follows:

$(*name*:*string1*=*string2*)

where *string1* is either a suffix, or a word to be replaced in the macro definition, and *string2* is the replacement suffix or word. Words in a macro value are separated by SPACE, TAB, and escaped NEWLINE characters.

*Pattern Replacement Macro References*

Pattern matching replacements can also be applied to macros, with a reference of the form:

$(*name*: *op*%*os*= *np*%*ns*)

where *op* is the existing (old) prefix and *os* is the existing (old) suffix, *np* and *ns* are the new prefix and new suffix, respectively, and the pattern matched by % (a string of zero or more characters), is carried forward from the value being replaced. For example:

**PROGRAM=fabricate**
**DEBUG= $(PROGRAM:%=tmp/%-g)**

sets the value of **DEBUG** to **tmp/fabricate-g.**

Note: pattern replacement macro references cannot be used in the dependency list of a pattern matching rule; the % characters are not evaluated independently. Also, any number of % metacharacters can appear after the equal-sign.

*Appending to a Macro*

Words can be appended to macro values as follows:

*macro* += *word* ...

**Special-Purpose Macros**

When the **MAKEFLAGS** variable is present in the environment, **make** takes options from it, in combination with options entered on the command line. **make** retains this combined value as the **MAKEFLAGS** macro, and exports it automatically to each command or shell it invokes.

Note: flags passed by way of **MAKEFLAGS** are only displayed when the **-d**, or **-dd** options are in effect.

The **MAKE** macro is another special case. It has the value **make** by default, and temporarily overrides the **-n** option for any line in which it is referred to. This allows nested invocations of **make** written as:

**$(MAKE)** ...

to run recursively, with the **-n** flag in effect for all commands but **make.** This lets you use '**make -n**' to test an entire hierarchy of makefiles.

For compatibility with the 4.2 BSD **make**, the **MFLAGS** macro is set from the **MAKEFLAGS** variable by prepending a '-'. **MFLAGS** is not exported automatically.

The **SHELL** macro, when set to a single-word value such as **/usr/bin/csh**, indicates the name of an alternate shell to use. The default is **/bin/sh.** Note: **make** executes commands that contain no shell metacharacters itself. Built-in commands, such as **dirs** in the C shell, are not recognized unless the command line includes a metacharacter (for instance, a semicolon). This macro is neither imported from, nor exported to the environment, regardless of **-e**. To be sure it is set properly, you must define this macro within every makefile that requires it.

The **KEEP_STATE** environment variable has the same effect as the **.KEEP_STATE:** special-function target. It enables command dependencies, hidden dependencies and writing of the state file.

The following macros are provided for use with cross-compilation:

**HOST_ARCH**
: The machine architecture of the host system. By default, this is the output of the **arch**(1) command prepended with '—'. Under normal circumstances, this value should never be altered by the user.

**TARGET_ARCH**
: The machine architecture of the target system. By default, the output of **arch**, prepended with '—'.

**HOST_MACH**
: The machine architecture of the host system. By default, this is the output of the **mach**(1), prepended with '—'. Under normal circumstances, this value should never be altered by the user.

**TARGET_MACH**

The machine architecture of the target system. By default, the output of **mach**, prepended with '—'.

**Dynamic Macros**

There are several dynamically maintained macros that are useful as abbreviations within rules. They are shown here as references; if you were to define them, **make** would simply override the definition.

**$*** The basename of the current target, derived as if selected for use with an implicit rule.

**$<** The name of a dependency file, derived as if selected for use with an implicit rule.

**$@** The name of the current target. This is the only dynamic macro whose value is strictly determined when used in a dependency list. (In which case it takes the form '$$@'.)

**$?** The list of dependencies that are newer than the target. Command-dependency checking is automatically suppressed for lines that contain this macro, just as if the command had been prefixed with a '?'. See the description of '?', under **Makefile Special Tokens**, above. You can force this check with the ! command-line prefix.

**$%** The name of the library member being processed. (See **Library Maintenance**, below.)

To refer to the $@ dynamic macro within a dependency list, precede the reference with an additional '$' character (as in, '$$@'). Because **make** assigns $< and $* as it would for implicit rules (according to the suffixes list and the directory contents), they may be unreliable when used within explicit target entries.

These macros can be modified to apply either to the filename part, or the directory part of the strings they stand for, by adding an upper case **F** or **D**, respectively (and enclosing the resulting name in parentheses or braces). Thus, '$(@D)' refers to the directory part of the string '$@'; if there is no directory part, '.' is assigned. $(@F) refers to the filename part.

**Conditional Macro Definitions**

A macro definition of the form:

*target-list* := *macro* = *value*

indicates that when processing any of the targets listed *and their dependencies* , *macro* is to be set to the *value* supplied. Note that if a conditional macro is referred to in a dependency list, the $ must be delayed (use $$ instead). Also, *target-list* may contain a % pattern, in which case the macro will be conditionally defined for all targets encountered that match the pattern. A pattern replacement reference can be used within the *value*.

You can temporarily append to a macro's value with a conditional definition of the form:

*target-list* := *macro* += *value*

**Predefined Macros**

**make** supplies the macros shown in the table that follows for compilers and their options, host architectures, and other commands. Unless these macros are read in as environment variables, their values are not exported by **make**. If you run **make** with any of these set in the environment, it is a good idea to add commentary to the makefile to indicate what value each is expected to take. If **–r** is in effect, **make** does not read the default makefile (**./default.mk** or **<make/default.mk>**) in which these macro definitions are supplied.

| *Table of Predefined Macros* | | |
|---|---|---|
| *Use* | *Macro* | *Default Value* |
| *Library Archives* | AR | ar |
| | ARFLAGS | rv |
| *Assembler Commands* | AS | as |
| | ASFLAGS | |
| | COMPILE.s | $(AS) $(ASFLAGS) $(TARGET_MACH) |
| | COMPILE.S | $(CC) $(ASFLAGS) $(CPPFLAGS) $(TARGET_MACH) -c |
| *C Compiler Commands* | CC | cc |
| | CFLAGS | |
| | CPPFLAGS | |
| | COMPILE.c | $(CC) $(CFLAGS) $(CPPFLAGS) -target $(TARGET_ARCH:-%=%) -c |
| | LINK.c | $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) -target $(TARGET_ARCH:-%=%) |
| *FORTRAN 77 Compiler Commands* | FC | f77 |
| | FFLAGS | |
| | COMPILE.f | $(FC) $(FFLAGS) $(TARGET_ARCH) -c |
| | LINK.f | $(FC) $(FFLAGS) $(TARGET_ARCH) $(LDFLAGS) |
| | COMPILE.F | $(FC) $(FFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c |
| | LINK.F | $(FC) $(FFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH) |
| *Link Editor Command* | LD | ld |
| | LDFLAGS | |
| *lex Command* | LEX | lex |
| | LFLAGS | |
| | LEX.l | $(LEX) $(LFLAGS) -t |
| *lint Command* | LINT | lint |
| | LINTFLAGS | |
| | LINT.c | $(LINT) $(LINTFLAGS) $(CPPFLAGS) $(TARGET_ARCH) |
| *Modula 2 Commands* | M2C | m2c |
| | M2FLAGS | |
| | MODFLAGS | |
| | DEFFLAGS | |
| | COMPILE.def | $(M2C) $(M2FLAGS) $(DEFFLAGS) $(TARGET_ARCH) |
| | COMPILE.mod | $(M2C) $(M2FLAGS) $(MODFLAGS) $(TARGET_ARCH) |
| *Pascal Compiler Commands* | PC | pc |
| | PFLAGS | |
| | COMPILE.p | $(PC) $(PFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c |
| | LINK.p | $(PC) $(PFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH) |
| *Ratfor Compilation Commands* | RFLAGS | |
| | COMPILE.r | $(FC) $(FFLAGS) $(RFLAGS) $(TARGET_ARCH) -c |
| | LINK.r | $(FC) $(FFLAGS) $(RFLAGS) $(TARGET_ARCH) $(LDFLAGS) |
| *rm Command* | RM | rm -f |
| *sccs Command* | SCCSFLAGS | |
| | SCCSGETFLAGS | -s |
| *yacc Command* | YACC | yacc |
| | YFLAGS | |
| | YACC.y | $(YACC) $(YFLAGS) |
| *Suffixes List* | SUFFIXES | .o .c .c~ .s .s~ .S .S~ .ln .f .f~ .F .F~ .l .l~ .mod .mod~ .sym .def .def~ .p .p~ .r .r~ .y .y~ .h .h~ .sh .sh~ .cps .cps~ |

**Implicit Rules**

When a target has no entry in the makefile, **make** attempts to determine its class (if any) and apply the rule for that class. An implicit rule describes how to build any target of a given class, from an associated dependency file. The class of a target can be determined either by a pattern, or by a suffix; the corresponding dependency file (with the same basename) from which such a target might be built. In addition to a predefined set of implicit rules, make allows you to define your own, either by pattern, or by suffix.

*Pattern Matching Rules*

A target entry of the form:

*tp%ts*: *dp%ds*
*rule*

is a pattern matching rule, in which *tp* is a target prefix, *ts* is a target suffix, *dp* is a dependency prefix, and *ds* is a dependency suffix (any of which may be null). The % stands for a basename of zero or more characters that is matched in the target, and is used to construct the name of a dependency. When **make** encounters a match in its search for an implicit rule, it uses the rule in that target entry to build the target from the dependency file. Pattern-matching implicit rules typically make use of the $@ and $< dynamic macros as placeholders for the target and dependency names. Other, regular dependencies may occur in the dependency list. An entry of the form:

*tp%ts*: [*dependency*...] *dp%ds* [*dependency*...]
*rule*

is a valid pattern matching rule.

*Suffix Rules*

When no pattern matching rule applies, **make** checks the target name to see if it ends with a suffix in the known suffixes list. If so, **make** checks for any suffix rules, as well as a dependency file with same root and another recognized suffix, from which to build it.

The target entry for a suffix rule takes the form:

*DsTs*:
*rule*

where *Ts* is the suffix of the target, *Ds* is the suffix of the dependency file, and *rule* is the rule for building a target in the class. Both *Ds* and *Ts* must appear in the suffixes list. (A suffix need not begin with a '.' to be recognized.)

A suffix rule with only one suffix describes how to build a target having a null (or no) suffix from a dependency file with the indicated suffix. For instance, the **.c** rule could be used to build an executable program named **file** from a C source file named **'file.c'**. If a target with a null suffix has an explicit dependency, **make** omits the search for a suffix rule.

| *Table of Standard Implicit (Suffix) Rules* | | |
|---|---|---|
| *Use* | *Implicit Rule Name* | *Command Line* |
| *Assembly Files* | .s.o | $(COMPILE.s) –o $@ $< |
| | .s.a | $(COMPILE.s) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |
| | .S.o | $(COMPILE.S) –o $@ $< |
| | .S.a | $(COMPILE.S) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |
| *C Files* | .c | $(LINK.c) –o $@ $< $(LDLIBS) |
| | .c.ln | $(LINT.c) $(OUTPUT_OPTION) –i $< |
| | .c.o | $(COMPILE.c) $(OUTPUT_OPTION) $< |
| | .c.a | $(COMPILE.c) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |
| *FORTRAN 77 Files* | .f | $(LINK.f) –o $@ $< $(LDLIBS) |
| | .f.o | $(COMPILE.f) $(OUTPUT_OPTION) $< |
| | .f.a | $(COMPILE.f) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |
| | .F | $(LINK.F) –o $@ $< $(LDLIBS) |
| | .F.o | $(COMPILE.F) $(OUTPUT_OPTION) $< |
| | .F.a | $(COMPILE.F) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |
| *lex Files* | .l | $(RM) $*.c<br>$(LEX.l) $< > $*.c<br>$(LINK.c) –o $@ $*.c $(LDLIBS)<br>$(RM) $*.c |
| | .l.c | $(RM) $@<br>$(LEX.l) $< > $@ |
| | .l.ln | $(RM) $*.c<br>$(LEX.l) $< > $*.c<br>$(LINT.c) –o $@ –i $*.c<br>$(RM) $*.c |
| | .l.o | $(RM) $*.c<br>$(LEX.l) $< > $*.c<br>$(COMPILE.c) –o $@ $*.c<br>$(RM) $*.c |
| *Modula 2 Files* | .mod<br>.mod.o<br>.def.sym | $(COMPILE.mod) –o $@ –e $@ $<<br>$(COMPILE.mod) –o $@ $<<br>$(COMPILE.def) –o $@ $< |
| *NeWS* | .cps.h | cps $*.cps |
| *Pascal Files* | .p | $(LINK.p) –o $@ $< $(LDLIBS) |
| | .p.o | $(COMPILE.p) $(OUTPUT_OPTION) $< |
| *Ratfor Files* | .r | $(LINK.r) –o $@ $< $(LDLIBS) |
| | .r.o | $(COMPILE.r) $(OUTPUT_OPTION) $< |
| | .r.a | $(COMPILE.r) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |

| *Table of Standard Implicit (Suffix) Rules (continued)* | | |
|---|---|---|
| *Use* | *Implicit Rule Name* | *Command Line* |
| *SCCS Files* | **.SCCS_GET** | **sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@ -G$@** |
| *Shell Scripts* | **.sh** | **cat $< >$@**<br>**chmod +x $@** |
| *yacc Files* | **.y** | **$(YACC.y) $<**<br>**$(LINK.c) -o $@ y.tab.c $(LDLIBS)**<br>**$(RM) y.tab.c** |
| | **.y.c** | **$(YACC.y) $<**<br>**mv y.tab.c $@** |
| | **.y.ln** | **$(YACC.y) $<**<br>**$(LINT.c) -o $@ -i y.tab.c**<br>**$(RM) y.tab.c** |
| | **.y.o** | **$(YACC.y) $<**<br>**$(COMPILE.c) -o $@ y.tab.c**<br>**$(RM) y.tab.c** |

**make** reads in the standard set of implicit rules from the file **<make/default.mk>**, unless **-r** is in effect, or there is a **default.mk** file in the local directory that does not **include** that file.

**The Suffixes List**

The suffixes list is given as the list of dependencies for the '**.SUFFIXES:**' special-function target. The default list is contained in the **SUFFIXES** macro (See *Table of Predefined Macros* for the standard list of suffixes). You can define additional **.SUFFIXES:** targets; a **.SUFFIXES** target with no dependencies clears the list of suffixes. Order is significant within the list; **make** selects a rule that corresponds to the target's suffix and the first dependency-file suffix found in the list. To place suffixes at the head of the list, clear the list and replace it with the new suffixes, followed by the default list:

**.SUFFIXES:**
**.SUFFIXES:** *suffixes* **$(SUFFIXES)**

A tilde (~) indicates that if a dependency file with the indicated suffix (minus the ~) is under SCCS its most recent version should be retrieved, if necessary, before the target is processed.

**Library Maintenance**

A target name of the form:

*lib*(*member* ...)

refers to a member, or a space-separated list of members, in an **ar**(1V) library.

The dependency of the library member on the corresponding file must be given as an explicit entry in the makefile. This can be handled by a pattern matching rule of the form:

*lib*(%.*s*): %.*s*

where .*s* is the suffix of the member; this suffix is typically **.o** for object libraries.

A target name of the form

*lib*((*symbol*))

refers to the member of a randomized object library (see **ranlib**(1)) that defines the entry point named *symbol*.

**Command Execution**

Command lines are executed one at a time, *each by its own process or shell*. Shell commands, notably **cd**, are ineffectual across an unescaped NEWLINE in the makefile. A line is printed (after macro expansion) just before being executed. This is suppressed if it starts with a '**@**', if there is a '**.SILENT:**' entry in the makefile, or if **make** is run with the **–s** option. Although the **–n** option specifies printing without execution, lines containing the macro **$(MAKE)** are executed regardless, and lines containing the **@** special character are printed. The **–t** (touch) option updates the modification date of a file without executing any rules. This can be dangerous when sources are maintained by more than one person.

**Bourne Shell Constructs**

To use the Bourne shell **if** control structure for branching, use a command line of the form:

```
if expression ; \
then command ; \
        ... ; \
else ; \
        ... ; \
fi
```

Although composed of several input lines, the escaped NEWLINE characters insure that **make** treats them all as one (shell) command line.

To use the Bourne shell **for** control structure for loops, use a command line of the form:

```
for var in list ; \
        do command; \
        ... ;\
done
```

To refer to a shell variable, use an escaped dollar-sign (\$). This prevents expansion of the dollar-sign by **make**.

**Command Substitutions**

To incorporate the standard output of a shell command in a macro, use a definition of the form:

*MACRO* **:sh** =*command*

The command is executed only once, standard error output is discarded, and NEWLINE characters are replaced with SPACEs. If the command has a non-zero exit status, **make** halts with an error.

To capture the output of a shell command in a macro reference, use a reference of the form:

**$(***MACRO* **:sh)**

where *MACRO* is the name of a macro containing a valid Bourne shell command line. In this case, the command is executed whenever the reference is evaluated. As with shell command substitutions, the reference is replaced with the standard output of the command. If the command has a non-zero exit status, **make** halts with an error.

*Signals*

**INT** and **QUIT** signals received from the keyboard halt **make** and remove the target file being processed unless that target is in the dependency list for '**.PRECIOUS:**'.

## EXAMPLES

This makefile says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) along with a common file **incl.h**:

```
pgm: a.o b.o
        $(LINK.c) -o $@ a.o b.o
a.o: incl.h a.c
        cc -c a.c
b.o: incl.h b.c
        cc -c b.c
```

The following makefile uses implicit rules to express the same dependencies:

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o b.o: incl.h
```

## FILES

| | |
|---|---|
| **makefile** | |
| **Makefile** | current version(s) of **make** description file |
| **SCCS/s.makefile** | |
| **SCCS/s.Makefile** | SCCS history files for the above makefile(s) |
| **default.mk** | default file for user-defined targets, macros, and implicit rules |
| **<make/default.mk>** | makefile for standard implicit rules and macros (not read if **default.mk** is) |
| **.make.state** | state file in the local directory |

## SEE ALSO

**ar**(1V), **cc**(1V), **cd**(1), **sccs-get**(1), **lex**(1), **ranlib**(1), **passwd**(5)

*SunOS User's Guide: Doing More*
*Programming Utilities and Libraries*

## DIAGNOSTICS

**make** returns an exit status of **1** when it halts as a result of an error. Otherwise it returns an exit status of 0.

**Do not know how to make** *target*. **Stop.**
There is no makefile entry for *target*, and none of **make**'s implicit rules apply (there is no dependency file with a suffix in the suffixes list, or the target's suffix is not in the list).

*** *target* **removed.**
**make** was interrupted while building *target*. Rather than leaving a partially-completed version that is newer than its dependencies, **make** removes the file named *target*.

*** *target* **not removed.**
**make** was interrupted while building *target* and *target* was not present in the directory.

*** *target* **could not be removed,** *reason*
**make** was interrupted while building *target*, which was not removed for the indicated reason.

**Read of include file '***file***' failed**
The makefile indicated in an **include** directive was not found, or was inaccessible.

**Loop detected when expanding macro value '***macro***'**
A reference to the macro being defined was found in the definition.

**Could not write state file '***file***'**
You used the **.KEEP_STATE:** target, but do not have write permission on the state file.

*** **Error code** *n*
The previous shell command returned a nonzero error code.

**** signal message*

The previous shell command was aborted due to a signal. If '**– core dumped**' appears after the message, a **core** file was created.

**Conditional macro conflict encountered**

Displayed only when **–d** is in effect, this message indicates that two or more parallel targets currently being processed depend on a target which is built differently for each by virtue of conditional macros. Since the target cannot simultaneously satisfy both dependency relationships, it is conflicted.

## BUGS

Some commands return nonzero status inappropriately; to overcome this difficulty, prefix the offending command line in the rule with a '–'.

Filenames with the characters '=', ':', or '@', do not work.

You cannot build **file.o** from **lib(file.o)**.

Options supplied by **MAKEFLAGS** should be reported for nested **make** commands. Use the **–d** option to find out what options the nested command picks up from **MAKEFLAGS**.

This version of **make** is incompatible in certain respects with previous versions:

- The **–d** option output is much briefer in this version. **–dd** now produces the equivalent voluminous output.
- **make** attempts to derive values for the dynamic macros '**$***', '**$<**', and '**$?**', while processing explicit targets. It uses the same method as for implicit rules; in some cases this can lead either to unexpected values, or to an empty value being assigned. (Actually, this was true for earlier versions as well, even though the documentation stated otherwise.)
- **make** no longer searches the current directory for SCCS history files.
- Suffix replacement in macro references are now applied after the macro is expanded.

There is no guarantee that makefiles created for this version of **make** will work with earlier versions.

If there is no **default.mk** file in the current directory, and the file **<make/default.mk>** is missing, **make** stops before processing any targets. To force **make** to run anyway, create an empty **default.mk** file in the current directory.

Once a dependency is made, **make** assumes the dependency file is present for the remainder of the run. If a rule subsequently removes that file and future targets depend on its existence, unexpected errors may result.

When hidden dependency checking is in effect, the **$?** macro's value includes the names of hidden dependencies. This can lead to improper filename arguments to commands when **$?** is used in a rule.

Pattern replacement macro references cannot be used in the dependency list of a pattern matching rule.

Unlike previous versions, this version of **make** strips a leading '**./**' from the value of the '**$@**' dynamic macro.

With automatic SCCS retrieval, this version of **make** does not support tilde suffix rules.

The only dynamic macro whose value is strictly determined when used in a dependency list is **$@** (takes the form '**$$@**').

**make** invokes the shell with the **–e** (exit-on-errors) argument. Thus, with semicolon-separated command sequences, execution of the later commands depends on the success of the former. This is consistent with **make**'s behavior of halting immediately when a problem occurs, but cannot be inferred from the syntax of the rule alone.

## NAME

man – display reference manual pages; find reference pages by keyword

## SYNOPSIS

**man** [ – ] [ **–t** ] [ **–M** *path* ] [ **–T** *macro-package* ] [ [ *section* ] *title* ... ] ...
**man** [ **–M** *path* ] **–k** *keyword* ...
**man** [ **–M** *path* ] **–f** *filename* ...

## DESCRIPTION

**man** displays information from the reference manuals. It can display complete manual pages that you select by *title*, or one-line summaries selected either by *keyword* (**–k**), or by the name of an associated file (**–f**).

A *section*, when given, applies to the *titles* that follow it on the command line (up to the next *section*, if any). **man** looks in the indicated section of the manual for those *titles*. *section* is either a digit (perhaps followed by a single letter indicating the type of manual page), or one of the words **new**, **local**, **old**, or **public**. The abbreviations **n**, **l**, **o** and **p** are also allowed. If *section* is omitted, **man** searches all reference sections (giving preference to commands over functions) and prints the first manual page it finds. If no manual page is located, **man** prints an error message.

The reference page sources are typically located in the **/usr/man/man?** directories. Since these directories are optionally installed, they may not reside on your host; you may have to mount **/usr/man** from a host on which they do reside. If there are preformatted, up-to-date versions in corresponding **cat?** or **fmt?** directories, **man** simply displays or prints those versions. If the preformatted version of interest is out of date or missing, **man** reformats it prior to display. If directories for the preformatted versions are not provided, **man** reformats a page whenever it is requested; it uses a temporary file to store the formatted text during display.

If the standard output is not a terminal, or if the '–' flag is given, **man** pipes its output through **cat**(1V). Otherwise, **man** pipes its output through **more**(1) to handle paging and underlining on the screen.

## OPTIONS

**–t** **man** arranges for the specified manual pages to be **troff**ed to a suitable raster output device (see **troff**(1) or **vtroff**(1)). If both the – and **–t** flags are given, **man** updates the **troff**ed versions of each named *title* (if necessary), but does not display them.

**–M** *path*
Change the search path for manual pages. *path* is a colon-separated list of directories that contain manual page directory subtrees. For example, **/usr/man/u_man:/usr/man/a_man** makes **man** search in the standard System V locations. When used with the **–k** or **–f** options, the **–M** option must appear first. Each directory in the *path* is assumed to contain subdirectories of the form **man[1-8l-p]**.

**–T** *macro-package*
**man** uses *macro-package* rather than the standard **–man** macros defined in **/usr/lib/tmac/tmac.an** for formatting manual pages.

**–k** *keyword* ...
**man** prints out one-line summaries from the **whatis** database (table of contents) that contain any of the given *keywords*. The **whatis** database is created using the **catman**(8) command with the **–w** option.

**–f** *filename* ...
**man** attempts to locate manual pages related to any of the given *filenames*. It strips the leading pathname components from each *filename*, and then prints one-line summaries containing the resulting basename or names. This option also uses the **whatis** database.

## MANUAL PAGES

Manual pages are **troff**(1)/**nroff**(1) source files prepared with the **–man** macro package. Refer to **man**(7), or *Formatting Documents* for more information.

When formatting a manual page, **man** examines the first line to determine whether it requires special processing.

### Referring to Other Manual Pages

If the first line of the manual page is a reference to another manual page entry fitting the pattern:

**.so man***?*/ *sourcefile*

**man** processes the indicated file in place of the current one. The reference must be expressed as a pathname relative to the root of the manual page directory subtree.

When the second or any subsequent line starts with **.so**, **man** ignores it; **troff**(1) or **nroff**(1) processes the request in the usual manner.

### Preprocessing Manual Pages

If the first line is a string of the form:

'\" *X*

where *X* is separated from the '"' by a single SPACE and consists of any combination of characters in the following list, **man** pipes its input to **troff**(1) or **nroff**(1) through the corresponding preprocessors.

| | |
|---|---|
| **e** | **eqn**(1), or **neqn** for **nroff** |
| **r** | **refer**(1) |
| **t** | **tbl**(1) |
| **v** | **vgrind**(1) |

If **eqn** or **neqn** is invoked, it will automatically read the file **/usr/pub/eqnchar** (see **eqnchar**(7)). If **nroff**(1) is invoked, **col**(1V) is automatically used.

## ENVIRONMENT

| | |
|---|---|
| **MANPATH** | If set, its value overrides **/usr/man** as the default search path. (The **–M** flag, in turn, overrides this value.) |
| **PAGER** | A program to use for interactively delivering **man**'s output to the screen. If not set, '**more –s**' (see **more**(1)) is used. |
| **TCAT** | The name of the program to use to display **troff**ed manual pages. If not set, '**lpr –t**' (see **lpr**(1)) is used. |
| **TROFF** | The name of the formatter to use when the **–t** flag is given. If not set, **troff** is used. |

## FILES

| | |
|---|---|
| **/usr/[share]/man** | root of the standard manual page directory subtree |
| **/usr/[share]/man/man?/*** | unformatted manual entries |
| **/usr/[share]/man/cat?/*** | **nroff**ed manual entries |
| **/usr/[share]/man/fmt?/*** | **troff**ed manual entries |
| **/usr/[share]/man/whatis** | table of contents and keyword database |
| **/usr/[share]/lib/tmac/tmac.an** | standard **–man** macro package |
| **/usr/pub/eqnchar** | |

## SEE ALSO

**apropos**(1), **cat**(1V), **col**(1V), **eqn**(1), **lpr**(1), **more**(1), **nroff**(1), **refer**(1), **tbl**(1), **troff**(1), **vgrind**(1), **vtroff**(1), **whatis**(1), **eqnchar**(7), **man**(7), **catman**(8)

## NOTES

Because **troff** is not 8-bit clean, **man** has not been made 8-bit clean.

The **–f** and **–k** options use the **/usr/man/whatis** database, which is created by **catman**(8).

## BUGS

The manual is supposed to be reproducible either on a phototypesetter or on an ASCII terminal. However, on a terminal some information (indicated by font changes, for instance) is necessarily lost.

Some dumb terminals cannot process the vertical motions produced by the **e** (**eqn**(1)) preprocessing flag. To prevent garbled output on these terminals, when you use **e** also use **t**, to invoke **col**(1V) implicitly. This workaround has the disadvantage of eliminating superscripts and subscripts — even on those terminals that can display them. CTRL-Q will clear a terminal that gets confused by **eqn**(1) output.

**NAME**

mesg – permit or deny messages on the terminal

**SYNOPSIS**

**mesg [ n ] [ y ]**

**DESCRIPTION**

**mesg** with argument **n** forbids messages with **write**(1) by revoking non-user write permission on the user's terminal. **mesg** with argument **y** reinstates permission. All by itself, **mesg** reports the current state without changing it.

**FILES**

**/dev/tty***

**SEE ALSO**

**write**(1), **talk**(1)

**DIAGNOSTICS**

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

**NAME**

mkdir – make a directory

**SYNOPSIS**

**mkdir** [ **–p** ] *dirname*...

**DESCRIPTION**

**mkdir** creates directories. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

The **–p** flag allows missing parent directories to be created as needed.

With the exception of the **set-gid** bit, the current **umask**(2V) setting determines the mode in which directories are created. The new directory inherits the **set-gid** bit of the parent directory. Modes may be modified after creation by using **chmod**(1V).

**mkdir** requires write permission in the parent directory.

**SEE ALSO**

**chmod**(1V), **rm**(1), **mkdir**(2V), **umask**(2V)

## NAME

mkstr – create an error message file by massaging C source files

## SYNOPSIS

**mkstr** [ – ] *messagefile prefix filename...*

## DESCRIPTION

**mkstr** creates files of error messages. You can use **mkstr** to make programs with large numbers of error diagnostics much smaller, and to reduce system overhead in running the program — as the error messages do not have to be constantly swapped in and out.

**mkstr** processes each of the specified *filename*s, placing a massaged version of the input file in a file with a name consisting of the specified *prefix* and the original source file name. A typical example of using **mkstr** would be:

**mkstr pistrings processed *.c**

This command would cause all the error messages from the C source files in the current directory to be placed in the file **pistrings** and processed copies of the source for these files to be placed in files whose names are prefixed with *processed*.

To process the error messages in the source to the message file, **mkstr** keys on the string '**error("**' in the input stream. Each time it occurs, the C string starting at the '"' is placed in the message file followed by a null character and a NEWLINE character; the null character terminates the message so it can be easily used when retrieved, the NEWLINE character makes it possible to sensibly **cat** the error message file to see its contents. The massaged copy of the input file then contains a **lseek** pointer into the file which can be used to retrieve the message, that is:

```
        char efilname[ ] = "/usr/lib/pi_strings";
        int efil = -1;

        error(a1, a2, a3, a4)
        {
                char
                buf[256];
                if (efil < 0) {
                                efil = open(efilname, 0);
                                if (efil < 0) {
oops:
                                perror (efilname);
                                exit (1);
                }
        }
        if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
                goto oops;
        printf(buf, a2, a3, a4);
}
```

## OPTIONS

– Place error messages at the end of the specified message file for recompiling part of a large **mkstred** program.

## SEE ALSO

**xstr**(1)

## NAME

more, page – browse or page through a text file

## SYNOPSIS

**more** [ **–cdflsu** ] [ *–lines* ] [ *+linenumber* ] [ *+/pattern* ] [ *filename* ... ]

**page** [ **–cdflsu** ] [ *–lines* ] [ *+linenumber* ] [ *+/pattern* ] [ *filename* ... ]

## DESCRIPTION

**more** is a filter that displays the contents of a text file on the terminal, one screenful at a time. It normally pauses after each screenful, and prints **--More--** at the bottom of the screen. **more** provides a two-line overlap between screens for continuity. If **more** is reading from a file rather than a pipe, the percentage of characters displayed so far is also shown.

**more** scrolls up to display one more line in response to a RETURN character; it displays another screenful in response to a SPACE character. Other commands are listed below.

**page** clears the screen before displaying the next screenful of text; it only provides a one-line overlap between screens.

**more** sets the terminal to *noecho* mode, so that the output can be continuous. Commands that you type do not normally show up on your terminal, except for the / and ! commands.

If the standard output is not a terminal, **more** acts just like **cat**(1V), except that a header is printed before each file in a series.

## OPTIONS

**–c** Clear before displaying. Redrawing the screen instead of scrolling for faster displays. This option is ignored if the terminal does not have the ability to clear to the end of a line.

**–d** Display error messages rather than ringing the terminal bell if an unrecognized command is used. This is helpful for inexperienced users.

**–f** Do not fold long lines. This is useful when lines contain nonprinting characters or escape sequences, such as those generated when **nroff**(1) output is piped through **ul**(1).

**–l** Do not treat FORMFEED characters (CTRL-D) as "page breaks." If **–l** is not used, **more** pauses to accept commands after any line containing a ^L character (CTRL-D). Also, if a file begins with a FORMFEED, the screen is cleared before the file is printed.

**–s** Squeeze. Replace multiple blank lines with a single blank line. This is helpful when viewing **nroff**(1) output, on the screen.

**–u** Suppress generation of underlining escape sequences. Normally, **more** handles underlining, such as that produced by **nroff**(1), in a manner appropriate to the terminal. If the terminal can perform underlining or has a stand-out mode, **more** supplies appropriate escape sequences as called for in the text file.

*–lines* Display the indicated number of *lines* in each screenful, rather than the default (the number of lines in the terminal screen less two).

*+linenumber*
Start up at *linenumber*.

*+/pattern*
Start up two lines above the line containing the regular expression *pattern*. Note: unlike editors, this construct should *not* end with a '/'. If it does, then the trailing slash is taken as a character in the search pattern.

## USAGE

### Environment

**more** uses the terminal's **termcap**(5) entry to determine its display characteristics, and looks in the environment variable **MORE** for any preset options. For instance, to page through files using the **–c** mode by default, set the value of this variable to **–c**. (Normally, the command sequence to set up this environment variable is placed in the **.login** or **.profile** file).

### Commands

The commands take effect immediately; it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may type the line kill character to cancel the numerical argument being formed. In addition, the user may type the erase character to redisplay the '**--More--**(*xx*%)' message.

In the following commands, *i* is a numerical argument (**1** by default).

*i*SPACE Display another screenful, or *i* more lines if *i* is specified.

*i*RETURN Display another line, or *i* more lines, if specified.

*i*^**D** (CTRL–D) Display (scroll down) 11 more lines. *i* is given, the scroll size is set to *i*.

*i***d** Same as ^**D**.

*i***z** Same as SPACE, except that *i*, if present, becomes the new default number of lines per screenful.

*i***s** Skip *i* lines and then print a screenful.

*i***f** Skip *i* screenfuls and then print a screenful.

*i*^**B** (CTRL-B) Skip back *i* screenfuls and then print a screenful.

*b* Same as ^**B** (CTRL-D).

**q**
**Q** Exit from **more**.

**=** Display the current line number.

**v** Drop into the **vi**(1) editor at the current line of the current file.

**h** Help. Give a description of all the **more** commands.

*i*/*pattern* Search for the *i* th occurrence of the regular expression *pattern*. Display the screenful starting two lines prior to the line that contains the *i* th match for the regular expression *pattern*, or the end of a pipe, whichever comes first. If **more** is displaying a file and there is no such match, its position in the file remains unchanged. Regular expressions can be edited using erase and kill characters. Erasing back past the first column cancels the search command.

*i***n** Search for the *i* th occurrence of the last *pattern* entered.

' Single quote. Go to the point from which the last search started. If no search has been performed in the current file, go to the beginning of the file.

!*command* Invoke a shell to execute *command*. The characters % and !, when used within *command* are replaced with the current filename and the previous shell command, respectively. If there is no current filename, % is not expanded. Prepend a backslash to these characters to escape expansion.

*i***:n** Skip to the *i* th next filename given in the command line, or to the last filename in the list if *i* is out of range.

*i***:p** Skip to the *i* th previous filename given in the command line, or to the first filename if *i* is out of range. If given while **more** is positioned within a file, go to the beginning of the file. If **more** is reading from a pipe, **more** simply rings the terminal bell.

**:f** Display the current filename and line number.

**:q**
**:Q** Exit from **more** (same as **q** or **Q**).

**.** Dot. Repeat the previous command.

**^** Halt a partial display of text. **more** stops sending output, and displays the usual **--More--** prompt. Unfortunately, some output is lost as a result.

**FILES**

| | |
|---|---|
| **/etc/termcap** | terminal data base |
| **/usr/lib/more.help** | help file |

**SEE ALSO**

**cat**(1V), **csh**(1), **man**(1), **script**(1), **sh**(1), **environ**(5V), **termcap**(5)

**BUGS**

Skipping backwards is too slow on large files.

## NAME

mt – magnetic tape control

## SYNOPSIS

**mt** [ **–f** *tapename* ] *command* [ *count* ]

## DESCRIPTION

**mt** sends commands to a magnetic tape drive. If *tapename* is not specified, the environment variable **TAPE** is used. If **TAPE** does not exist, **mt** uses the device **/dev/rmt12**. *tapename* refers to a raw tape device. By default, **mt** performs the requested operation once; multiple operations may be performed by specifying *count*.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

**mt** returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized or if **mt** was unable to open the specified tape drive, and 2 if an operation failed.

## OPTIONS

**eof, weof** Write *count* EOF marks at the current position on the tape.

**fsf** Forward space over *count* EOF marks. The tape is positioned on the first block of the file.

**fsr** Forward space *count* records.

**bsf** Back space over *count* EOF marks. The tape is positioned on the beginning-of-tape side of the EOF mark.

**bsr** Back space *count* records.

**nbsf** Back space *count* files. The tape is positioned on the first block of the file. This is equivalent to *count+1* **bsf**'s followed by one **fsf**.

**asf** Absolute space to *count* file number. This is equivalent to a **rewind** followed by a **fsf** *count*.

For the following commands, *count* is ignored:

**eom** Space to the end of recorded media on the tape. This is useful for appending files onto previously written tapes.

**rewind** Rewind the tape.

**offline, rewoffl** Rewind the tape and, if appropriate, take the drive unit off-line by unloading the tape.

**status** Print status information about the tape unit.

**retension** Rewind the cartridge tape completely, then wind it forward to the end of the reel and back to beginning-of-tape to smooth out tape tension.

**erase** Erase the entire tape.

## FILES

| | |
|---|---|
| **/dev/rmt*** | magnetic tape interface |
| **/dev/rar*** | Archive cartridge tape interface |
| **/dev/rst*** | SCSI tape interface |

## SEE ALSO

**ar**(4S), **mtio**(4), **st**(4S), **tm**(4S), **xt**(4S) **environ**(5V)

## BUGS

Not all devices support all options. Some options are hardware-dependent. Refer to the corresponding device manual page.

**WARNINGS**

The **bsf** option for SCSI tape in SunOS 4.0.3 is incompatible with this release and releases prior to 4.0.3.

## NAME

mv – move or rename files

## SYNOPSIS

**mv** [ – ] [ –**fi** ] *filename1 filename2*
**mv** [ – ] [ –**fi** ] *directory1 directory2*
**mv** [ – ] [ –**fi** ] *filename ... directory*

## DESCRIPTION

**mv** moves files and directories around in the file system. A side effect of **mv** is to rename a file or directory. The three major forms of **mv** are shown in the synopsis above.

The first form of **mv** moves (changes the name of) *filename1* to *filename2*. If *filename2* already exists, it is removed before *filename1* is moved. If *filename2* has a mode which forbids writing, **mv** prints the mode (see **chmod**(2V)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place, otherwise **mv** exits.

The second form of **mv** moves (changes the name of) *directory1* to *directory2*, *only* if *directory2* does not already exist — if it does, the third form applies.

The third form of **mv** moves one or more *filename*s (may also be directories) with their original names, into the last *directory* in the list.

**mv** refuses to move a file or directory onto itself.

## OPTIONS

– Interpret all the following arguments to **mv** as file names. This allows file names starting with minus.

–**f** Force. Override any mode restrictions and the –**i** option. The –**f** option also suppresses any warning messages about modes which would potentially restrict overwriting.

–**i** Interactive mode. **mv** displays the name of the file or directory followed by a question mark whenever a move would replace an existing file or directory. If you type a line starting with **y**, **mv** moves the specified file or directory, otherwise **mv** does nothing with that file or directory.

## SEE ALSO

**cp**(1), **ln**(1V), **chmod**(2V), **rename**(2V)

## DIAGNOSTICS

**mv:** *pathname*: **rename: Permission denied**
Attempted to move *pathname* into a directory that did not have write permission.

## BUGS

If *filename1* and *filename2* are on different file systems, then **mv** must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Modification times may be different than expected when **mv** must copy the file's data, rather than simply updating a directory entry.

**mv** will not move a directory from one file system to another. Use **cp**(1) instead.

## NAME

nawk – pattern scanning and processing language

## SYNOPSIS

**awk** [ **–f** *program-file* ] [ **–F** *c* ] [ *program* ] [ *variable* =*value* ... ] [ *filename*... ]

## DESCRIPTION

**nawk** is a new version of **awk**(1) that provides additional features including, dynamic regular expressions, additional built-ins and operators, and user defined functions. Other implementations refer to this command by its original name, **awk**, choosing to replace the original program with the enhanced one. Since there is a slight incompatibility between the two versions (see BUGS below) both versions are available in the SunOS environment, the original, **awk**, and the enhanced, **nawk**.

**nawk** scans each input *filename* for lines that match any of a set of patterns specified in *program*. *program* string must be enclosed in single quotes (') to protect it from the shell. For each pattern in *program* there may be an associated action performed when a line of a *filename* matches the pattern. The set of pattern-action statements may appear literally as *program* or in a file specified with the **–f** *program-file* option.

## OPTIONS

**–f** *filename*

Specify the contents of *filename* as the source for the program.

**–F** *c* Set the input field separator to *c*. If the field separator is longer than one character, it is taken to be a regular expression, and should be enclosed in single quotes to protect special characters from the shell.

*variable*=*value*

Set a built-in variable to *value* before the first record of the next *filename* is read. See **Built-in Variables** below for a complete list of available variables.

## USAGE

### Input Lines

Input files are read in order; if there are no files, the standard input is read. The file name '–' means the standard input. Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is normally made up of fields separated by white space. This default can be changed by using the FS built-in variable or the **–F** *c* option.) The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

### Pattern-action Statements

**nawk** programs contain pattern-action statements of the form:

*pattern* { *action* }

Either pattern or action may be omitted. If there is no action with a pattern, the matching line is printed. If there is no pattern with an action, the action is performed on every input line.

Patterns are arbitrary Boolean combinations (**!**, **||**, **&&**, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

*expression relop expression*
*expression matchop regular expression*
*expression* **in** *array-name*
*(expression, expression, ...)* **in** *array-name*

where *relop* is any of the six relational operators in C, and *matchop* is either **~** (contains) or **!~** (does not contain). An expression is an arithmetic expression, a relational expression, the special expression

var **in** array,

or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line has been read and after the last input line has been read respectively. They are the only patterns that require an *action* statement. These keywords do not combine with any other patterns.

Regular expressions are as in **egrep** (see **grep**(1V). In patterns they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second pattern.

An action is a sequence of statements. A statement may be one of the following:

**if** ( *expression* ) *statement* [ **else** *statement* ]
**while** ( *expression* ) *statement*
**do** *statement* **while** ( *expression* )
**for** ( *expression* ; *expression* ; *expression* ) *statement*
**for** ( *var* **in** array ) *statement*
**delete** array[*subscript*]
**break**
**continue**
{ [ statement ] ... }
*expression* # **commonly variable** = *expression*
**print** [ *expression-list* ] [ >*expression* ]
**printf** *format* [ , *expression-list* ] [ >*expression* ]
**next** # skip remaining patterns on this input line
**exit** [*expr*] # skip the rest of the input; exit status is expr
**return** [*expr*]

Statements are terminated by semicolons, right braces, or NEWLINE characters. An empty expression-list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators +, −, *, /, %, and concatenation (indicated by a blank). The **C operators** ++, —, +=, −=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted *x*[*i*]), or fields. Variables are initialized to the null string or zero. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").

The **print** statement prints its arguments on the standard output, or on a file if >*expression* is present, or on a pipe if '| *cmd*' is present. The arguments are separated by the current output field separator and terminated by the output record separator. The **printf** statement formats its expression list according to the format (see **printf**(3V)).

**Built-in Variables**

A regular expression may be used to separate fields by using the **−F c** option or by assigning the expression to the built-in variable FS. The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

Built-in variables include:

| | |
|---|---|
| ARGC | Command line argument count. |
| ARGV | Command line argument array. |
| FILENAME | Name of the current input file. |
| FNR | Ordinal number of the current record in the current file. |
| FS | Input field separator regular expression (default blank). |
| NF | Number of fields in the current record. |
| NR | Ordinal number of the current record. |
| OFMT | Output format for numbers (default **%.6g**). |

| | |
|---|---|
| OFS | Output field separator (default blank). |
| ORS | Output record separator (default NEWLINE). |
| RS | Input record separator (default NEWLINE). |
| SUBSEP | Separates multiple subscripts (default is 034). |

**nawk** has a variety of built-in functions: arithmetic, string, input/output, and general.

The arithmetic functions are: *atan2*, *cos*, *exp*, *int*, *log*, *rand*, *sin*, *sqrt*, and *srand*. *int* truncates its argument to an integer. *rand* returns a random number between 0 and 1. *srand* ( expr ) sets the seed value for *rand* to *expr* or uses the time of day if *expr* is omitted.

The string functions are:

**gsub**(*for*,'*repl*,'*in*)
behaves like **sub** (see below), except that it replaces successive occurrences of the regular expression (like the **ed** global substitute command).

**index(s,t)** returns the position in string **s** where string **t** first occurs, or 0 if it does not occur at all.

**int** truncates to an integer value.

**length(s)** returns the length of its argument taken as a string, or of the whole line if there is no argument.

**match(s,'re)** returns the position in string **s** where the regular expression **re** occurs, or 0 if it does not occur at all. RSTART is set to the starting position (which is the same as the returned value), and RLENGTH is set to the length of the matched string.

**rand** random number on (0, 1).

**split**(*s*,'*a*,'*fs*) splits the string **s** into array elements *a*[*1*], *a*[*2*], *a*[*n*], and returns *n*. The separation is done with the regular expression **fs** or with the field separator FS if **fs** is not given.

**srand** sets the seed for **rand**

**sprintf**(*fmt*,'*expr*,'*expr*,'...)
formats the expressions according to the **printf**(3V) format given by **fmt** and returns the resulting string.

**sub**(*for*,'*repl*,'*in*) substitutes the string **repl** in place of the first instance of the regular expression **for** in string **in** and returns the number of substitutions. If **in** is omitted, **nawk** substitutes in the current record ($0).

**substr**(*s*,'*m*,'*n*) returns the *n*-character substring of **s** that begins at position *m*. The input/output and general functions are:

**close(filename)** closes the file or pipe named *filename*.

*cmd* | **getline** pipes the output of *cmd* into **getline**; each successive call to **getline** returns the next line of output from *cmd*.

**getline** sets **$0** to the next input record from the current input file.

**getline <file** sets **$0** to the next record from *file*.

**getline x** sets variable **x** instead.

**getline x <file** sets **x** from the next record of *file*.

*system*(*cmd*) executes **cmd** and returns its exit status. All forms of **getline** return 1 for successful input, 0 for end of file, and −1 for an error.

**nawk** also provides user-defined functions. Such functions may be defined (in the pattern position of a pattern-action statement) as

*function name*(*args*,...) { *stmts* }
*func name*(*args*,...) { *stmts* }

Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The **return** statement may be used to return a value.

**EXAMPLES**

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
        { print $2, $1 }
```

Add up first column, print sum and average:

```
{ s += $1 }
END     { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; —i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Simulate **echo**(1V):

```
BEGIN {
        for (i = 1; i < ARGC; i++)
                printf "%s", ARGV[i]
        printf "\n"
        exit
}
```

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
    { print }
```

```
example% nawk –f program n=5 input
```

**SEE ALSO**

**grep**(1V), **lex**(1), **sed**(1V), **printf**(3V)

*Editing Text Files*

A. V. Aho, B. W. Kerninghan, P. J. Weinberger, *The AWK Programming Language* Addison-Wesley, 1988.

**BUGS**

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

Pattern-action statements must be separated by either a semi-colon or a NEWLINE. This is an incompatibility with the old version of **awk**.

## NAME

newgrp – log in to a new group

## SYNOPSIS

**newgrp** [ – ] [ **group** ]

## DESCRIPTION

**newgrp** changes a user's group identification. Only the group-ID is changed; the user remains a member of all groups previously established by **setgroups** (see **getgroups**(2V)). The user remains logged in and the current directory is unchanged, but the group-ID of newly-created files will be set to the new effective group-ID (see **open**(2V)). The user is always given a new shell, replacing the current shell, regardless of whether **newgrp** terminated successfully or due to an error condition (such as an unknown group).

Exported variables retain their values after invoking **newgrp**; however, all unexported variables are either reset to their default value or set to null. System variables (such as **HOME, LOGNAME, PATH, SHELL, TERM,** and **USER**), unless exported by the system or explicitly exported by the user, are reset to default values. Note: the shell command **export** (see **sh**(1)) is the method to export variables, while the C shell command **setenv** (see **csh**(1)) implicitly exports its argument.

With no arguments, **newgrp** changes the group identification back to the group specified in the user's password file entry.

If the first argument to **newgrp** is a '–', the environment is changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

## FILES

| | |
|---|---|
| **/etc/group** | system group file |
| **/etc/passwd** | system password file |

## SEE ALSO

**csh**(1), **login**(1), **sh**(1), **su**(1V), **open**(2V), **getgroups**(2V), **initgroups**(3), **environ**(5V), **group**(5), **passwd**(5)

## NOTES

For consistency with **login** naming rules (which do not allow 8-bit file names), group identifications cannot contain 8-bit characters. See **login**(1) for explanations about why **login** is not 8-bit clean.

## BUGS

There is no convenient way to enter a password into **/etc/group**. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

## NAME

nice – run a command at low priority

## SYNOPSIS

**nice** [ *–number* ] *command* [ *arguments* ]

## DESCRIPTION

There are two distinct versions of **nice**: it is built in to the C shell, and is an executable program available in **/usr/bin/nice** for use with the Bourne shell.

**nice** executes *command* with the "nice" value *number*. The nice value is one of the factors used by the kernel to determine a process's scheduling priority. Scheduling priorities range from 0 to 127. The higher the value, the lower the command's scheduling priority, and the lower the value, the higher the command's scheduling priority. In addition to the nice value, the kernel also recent CPU usage by the process, the time the process has been waiting to run, and other factors to arrive at scheduling priority.

If the *number* argument is present, the nice value is incremented or decremented by that amount, between the limits –20 and 19. If there is no *number* argument, the default nice value is 10 for the Bourne shell, and 4 for the C-shell.

The super-user may run commands with priority higher than normal by using negative nice values, such as **–10**.

## EXAMPLES

The following examples illustrate the use of nice values for users (not the super-user) using **/usr/bin/nice**. The examples use the **–l** option to **ps**(1) because it shows both the nice value and the kernel scheduling priority. Notice the **NI** and **PRI** columns. In the first example, the user doesn't use **nice**, so the niceness is zero, the default value, which is reflected by 0 in the **NI** column. The corresponding process scheduling priority is shown in the **PRI** column as 28 (this may vary because of the other factors the kernel's scheduler uses).

In the second example, the user uses a nice value of 10, and the corresponding priority is 53, a higher numerical value but a lower priority. Notice that this is the same as:

example% **nice ps –l**

because the default nice value is 10.

In the third example, the user asks that the nice value be incremented by 20, but it's shown as 19 under **NI**, because that's the upper limit of niceness.

A fourth example shows the error message when an ordinary user tries to decrement the nice value.

```
example% ps –l
   F UID   PID  PPID CP PRI NI  SZ  RSS WCHAN   STAT TT  TIME COMMAND
   ...
   19442 16623  9725 12  28  0 120  336       R    p2  0:00 ps –l
   ...
example% nice –10 ps –l
   F UID   PID  PPID CP PRI NI  SZ  RSS WCHAN   STAT TT  TIME COMMAND
   ...
   19442 16608 16606 32  53 10 120  328       R N  p2  0:00 ps –l
   ...
example% nice –20 ps –l
   F UID   PID  PPID CP PRI NI  SZ  RSS WCHAN   STAT TT  TIME COMMAND
   ...
   19442 16609 16606 37  72 19 120  328       R N  p2  0:00 ps –l
   ...
```

```
example% nice —20 ps –l
nice: setpriority: Permission denied
example%
```

**SEE ALSO**

**csh**(1), **getpriority**(2), **nice**(3V), **pstat**(8), **renice**(8)

**DIAGNOSTICS**

**nice** returns the exit status of the subject command.

**BUGS**

The **nice** command has a different syntax than the **/usr/bin/nice** command described here. It uses the plus sign, +, to increment nice values, for example:

**example% nice** +*number*

increments the nice value by *number*. It uses a single a minus sign, –, to decrement nice values for super-user.

## NAME

**nl** – line numbering filter

## SYNOPSIS

**nl** [ **–p** ] [ **–h***type* ] [ **–b***type* ] [ **–f***type* ] [ **–v***start* ] [ **–i***incr* ] [ **–l***num* ] [ **–s***sep* ] [ **–w***width* ] [ **–n***fmt* ] [ **–d***delim* ] *filename*

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**nl** reads lines from *filename* (or the standard input), numbers them according to the options in effect, and sends its output to the standard output.

**nl** views the text it reads in terms of logical pages. Line numbering is normally reset at the start of each page. A logical page is composed of header, body and footer sections. The start of each page section is signaled by input lines containing section delimiters only:

*Start of file*

\:\:\:
*header*

\:\:
*body*

\:
*footer*

Empty sections are valid. Different line-numbering options are available within each section. The default scheme is no numbering for headers and footers.

## OPTIONS

**–p** Do not restart numbering at logical page delimiters.

**–b***type* Specify which logical page body lines are to be numbered. *type* is one of:
- **a** number all lines
- **t** number lines with printable text only (the default)
- **n** no line numbering
- **p** *rexp* number only lines that contain the regular expression *rexp*

**–h***type* Same as **–b***type* except for the header. The default *type* for the logical page header is **n** (no lines numbered).

**–f***type* Same as **–b***type* except for the footer. The default for logical page footer is **n** (no lines numbered).

**–v***start* *start* is the initial value used to number logical page lines. The default is 1.

**–i***incr* *incr* is the increment by which to number logical page lines. The default is 1.

**–s***sep* *sep* is the character(s) used to separate the line number from the corresponding text line. The default is a TAB.

**–w***width* *width* is the number of characters to be used for the line-number field. The default is 6.

**–n***fmt* **fmt** is the line numbering format. Recognized values are:
- **rn** right justified, leading zeroes suppressed (the default)
- **ln** left justified, leading zeroes suppressed
- **rz** right justified, leading zeroes kept

**–l***num* *num* is the number of blank lines to be considered as one. For example, **–l2** results in only the second adjacent blank being numbered (if the appropriate **–ha**, **–ba**, and/or **–fa** option is set). The default is 1.

**–d***xx* The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **–d** and the delimiter characters. To enter a backslash, use two backslashes.

**EXAMPLE**

The command:

**nl –v10 –i10 –d!+** *filename1*

will number *filename1* starting at line number 10 with an increment of ten. The logical page delimiters are **!+**.

**SEE ALSO**

**pr**(1V)

## NAME

nm – print symbol name list

## SYNOPSIS

**nm** [ **–gnoprsua** ] [ [ *filename* ] ...

## DESCRIPTION

**nm** prints the name list (symbol table) of each object *filename* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *filename* is given, the symbols in **a.out** are listed.

### Output Format

Each symbol name is preceded by its value (blanks if undefined) and one of the letters:

| | |
|---|---|
| **A** | absolute |
| **B** | bss segment symbol |
| **C** | common symbol |
| **D** | data segment symbol |
| **f** | filename |
| **t** | a static function symbol |
| **T** | text segment symbol |
| **U** | undefined |
| – | debug, giving symbol table entries (see **–a** below) |

The type letter is upper case if the symbol is external, and lower case if it is local. The output is sorted alphabetically.

## OPTIONS

| | |
|---|---|
| **–a** | Print all symbols. |
| **–g** | Print only global (external) symbols. |
| **–n** | Sort numerically rather than alphabetically. |
| **–o** | Prepend file or archive element name to each output line rather than only once. |
| **–p** | Do not sort; print in symbol-table order. |
| **–r** | Sort in reverse order. |
| **–s** | Sort according to the size of the external symbol (computed from the difference between the value of the symbol and the value of the symbol with the next higher value). This difference is the value printed. |
| **–u** | Print only undefined symbols. |

## EXAMPLE

**example% nm**

prints the symbol list of the file named **a.out**, the default output file for the **C**, compiler.

## SEE ALSO

**ar**(1V), **as**(1), **cc**(1V), **ld**(1), **tmpnam**(3S), **a.out**(5), **ar**(5), **coff**(5)

## BUGS

To see what is in a shared library, run **nm** on the *static* (**.a**) version.

### Sun386i BUGS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **–v** and **–n** options should be used only in conjunction with the **–e** option.

**NAME**

nohup – run a command immune to hangups and quits

**SYNOPSIS**

**/usr/bin/nohup** *command* [ *arguments* ]

**SYSTEM V SYNOPSIS**

**/usr/5bin/nohup** *command* [ *arguments* ]

**AVAILABILITY**

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

There are three distinct versions of **nohup**: it is built in to the C shell, and is an executable program available in **/usr/bin/nohup** and **/usr/5bin/nohup** when using the Bourne shell.

The Bourne shell version of **nohup** executes *command* such that it is immune to HUP (hangup) and TERM (terminate) signals. If the standard output is a terminal, it is redirected to the file **nohup.out**. The standard error is redirected to follow the standard output.

The priority is incremented by 5. **nohup** should be invoked from the shell with '&' in order to prevent it from responding to interrupts or input from the next user.

**SYSTEM V DESCRIPTON**

Processes run by **nohup** are immune to HUP (hangup) and QUIT (quit) signals; **nohup** does not arrange to make them immune to a TERM (terminate) signal, so unless they arrange to be immune to a TERM signal, or the shell makes them immune to a TERM signal, they will receive that signal. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**. If the standard error is a terminal, it is redirected to the standard output, otherwise it is not redirected. The priority of the process run by **nohup** is not altered.

**EXAMPLE**

It is frequently desirable to apply **nohup** to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell script. The command

**example% nohup sh script**

applies to everything in **script**. (If the script is to be executed often, then the need to type *sh* can be eliminated by giving **script** execute permission). Add an ampersand and the contents of **script** are run in the background with interrupts also ignored (see **sh**(1)):

**example% nohup script &**

**FILES**

**nohup.out**
**$HOME/nohup.out**

**SEE ALSO**

**chmod**(1V), **csh**(1), **nice**(1), **sh**(1), **signal**(3V)

**BUGS**

If you use **csh**(1), then commands executed with '&' are automatically immune to HUP signals while in the background.

There is a C shell built-in command **nohup** that provides immunity from terminate, but does not redirect output to **nohup.out**.

**nohup** does not recognize command sequences. For instance,

**nohup** *command1* ; *command2*

applies only to *command1* and the command:

**nohup** (*command1* ; *command2*)

is syntactically incorrect.

Be careful of where the standard error is redirected. The following command may put error messages on tape, making it unreadable:

**nohup cpio −o < list > /dev/rmt/1m&**

while

**nohup cpio −o < list > /dev/rmt/1m 2>errors&**

puts the error messages into the file **errors**.

## NAME

nroff – format documents for display or line-printer

## SYNOPSIS

**nroff** [ **–ehig** ] [ **–m***name* ] [ **–n***N* ] [ **–o***pagelist* ] [ **–r***aN* ] [ **–s***N* ] [ **–T***name* ]

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**nroff** formats text in the named *files* for typewriter-like devices. See also **troff**(1). The full capabilities of **nroff** and **troff** are described in *Formatting Documents*.

If no *file* argument is present, **nroff** reads the standard input. An argument consisting of a '–' is taken to be a file name corresponding to the standard input.

## OPTIONS

Options may appear in any order so long as they appear *before* the files.

**–e** Produce equally-spaced words in adjusted lines, using full terminal resolution.

**–h** Use output TAB characters during horizontal spacing to speed output and reduce output character count. TAB settings are assumed to be every 8 nominal character widths.

**–i** Read the standard input after the input files are exhausted.

**–q** Invoke the simultaneous input-output mode of the **rd** request.

**–m***name*
Prepend the macro file **/usr/share/lib/tmac/tmac.name** to the input files.

**–n***N* Number first generated page *N*.

**–o***pagelist*
Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end.

**–r***aN* Set register *a* (one-character) to *N*.

**–s***N* Stop every *N* pages. **nroff** will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a NEWLINE.

**–T***name*
Prepare output for a device of the specified *name*. Known *name*s are:

| | |
|---|---|
| **37** | Teletype Corporation Model 37 terminal — this is the default. |
| **crt \| lpr \| tn300** | GE TermiNet 300, or any line printer or terminal without half-line capability. |
| **300** | DASI-300. |
| **300-12** | DASI-300 — 12-pitch. |
| **300S \| 302 \| dtc** | DASI-300S. |
| **300S-12 \| 302-12 \| dtc12** | DASI-300S. |
| **382** | DASI-382 (fancy DTC 382). |
| **382-12** | DASI-82 (fancy DTC 382 — 12-pitch). |
| **450 \| ipsi** | DASI-450 (Diablo Hyterm). |
| **450-12 \| ipsi12** | DASI-450 (Diablo Hyterm) — 12-pitch. |
| **450-12-8** | DASI-450 (Diablo Hyterm) — 12-pitch and lines-per-inch. |
| **450X** | DASI-450X (Diablo Hyterm). |
| **832** | AJ 832. |

| | |
|---|---|
| **833** | AJ 833. |
| **832-12** | AJ 832 — 12-pitch. |
| **833-12** | AJ 833 — 12-pitch. |
| **epson** | Epson FX80. |
| **itoh** | C:ITOH Prowriter. |
| **itoh-12** | C:ITOH Prowriter — 12-pitch. |
| **nec** | NEC 55?0 or NEC 77?0 Spinwriter. |
| **nec12** | NEC 55?0 or NEC 77?0 Spinwriter — 12-pitch. |
| **nec-t** | NEC 55?0/77?0 Spinwriter — Tech-Math/Times-Romanthimble. |
| **qume** | Qume Sprint — 5 or9. |
| **qume12** | Qume Sprint — 5 or 9,12-pitch. |
| **xerox** | Xerox 17?0 or Diablo 16?0. |
| **xerox12** | Xerox 17?0 or Diablo 16?0 — 12-pitch. |
| **x-ecs** | Xerox/Diablo 1730/630 — Extended Character Set. |
| **x-ecs12** | Xerox/Diablo 1730/630 — Extended Character Set, 12-pitch. |
| **man** | Dumb terminal, used for online man pages. |

## EXAMPLE

The following command:

**example% nroff -s4 -me users.guide**

formats **users.guide** using the **-me** macro package, and stopping every 4 pages.

## FILES

| | |
|---|---|
| **/tmp/ta*** | temporary file |
| **/usr/share/lib/tmac/tmac.*** | standard macro files |
| **/usr/share/lib/term/*** | terminal driving tables for **nroff** |
| **/usr/share/lib/term/README** | index to terminal description files |

## SEE ALSO

**checknr**(1), **col**(1V), **eqn**(1), **tbl**(1), **troff**(1), **term**(5), **man**(7), **me**(7), **ms**(7)

*Formatting Documents*

## NOTES

**nroff** is not 8-bit clean because making **nroff** 8-bit clean would require rewriting the **nroff** internals and filters. Also, some **nroff** syntax is based on ASCII only and does not lend itself to 8-bit character sequences.

## NAME

objdump – dump selected parts of a COFF object file

## SYNOPSIS

**objdump** [ *option* [ *modifier* ... ] ] *filename* ...

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

The **objdump** command dumps selected parts of each of its object *filename* arguments. This command is compatible with System V in all its options. It will accept both object files and archives of object files.

**objdump** command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

## OPTIONS

**–a** Dump the archive header of each member of each archive file argument.

**–g** Dump the global symbols in the symbol table of an archive.

**–f** Dump each file header.

**–o** Dump each optional header.

**–h** Dump section headers.

**–s** Dump section contents.

**–r** Dump relocation information.

**–l** Dump line number information.

**–t** Dump symbol table entries.

**–z** *name* Dump line number entries for the named function.

**–c** Dump the string table.

**–L** Interpret and print the contents of the **.lib** sections.

### Modifiers

The following *modifiers* are used in conjunction with the options listed above.

**–d** *number* Dump the section number, *number*, or the range of sections starting at *number* and ending at the *number* specified by **+d**.

**+d** *number* Dump sections in the range either beginning with first section or beginning with section specified by **–d**.

**–n** *name* Dump information pertaining only to the named entity. This *modifier* applies to **–h**, **–s**, **–r**, **–l**, and **–t**.

**–p** Suppress printing of the headers.

**–t** *index* Dump only the indexed symbol table entry. The **–t** used in conjunction with **+t**, specifies a range of symbol table entries.

**+t** *index* Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **–t** option.

**–u** Underline the name of the file for emphasis.

**–v** Dump information in symbolic representation rather than numeric (e.g., C_STATIC instead of **0X02**). This *modifier* can be used with all the above options except **–s** and **–o**.

**−z** *name* **,***number*
**−z** *name number*
Dump line number entry or range of line numbers starting at *number* for the named function.

**+z** *number* Dump line numbers starting at either function *name* or *number* specified by **−z**, up to *number* specified by **+z**.

White space separating an *option* and its *modifier* is optional.

**SEE ALSO**

**ar**(5), **coff**(5)

## NAME

od – octal, decimal, hexadecimal, and ascii dump

## SYNOPSIS

**od** [ *–format* ] [ *filename* ] [ [+]*offset*[.] [**b**] [*label*] ]

## SYSTEM V SYNOPSIS

**/usr/5bin/od** [*arguments*]

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**od** displays *filename*, or its standard input, in one or more dump formats as selected by the first argument. If the first argument is missing, **–o** (octal) is the default. Dumping continues until an EOF.

### Format Arguments

The meanings of the format argument characters are:

**a** Interpret bytes as characters and display them with their ASCII names. If the **p** character is given also, bytes with even parity are underlined. If the **P** character is given, bytes with odd parity are underlined. Otherwise the parity bit is ignored.

**b** Interpret bytes as unsigned octal.

**c** Interpret bytes as ASCII characters. Certain non-graphic characters appear as C escapes: NUL=**\0**, BACKSPACE=**\b**, FORMFEED=**\f**, NEWLINE=**\n**, RETURN=**\r**, TAB=**\t**; others appear as 3-digit octal numbers. Bytes with the parity bit set are displayed in octal.

**d** Interpret (short) words as unsigned decimal.

**f** Interpret long words as floating point.

**h** Interpret (short) words as unsigned hexadecimal.

**i** Interpret (short) words as signed decimal.

**l** Interpret long words as signed decimal.

**o** Interpret (short) words as unsigned octal.

**s**[*n*] Look for strings of ASCII graphic characters, terminated with a null byte. *n* specifies the minimum length string to be recognized. By default, the minimum length is 3 characters.

**v** Show all data. By default, display lines that are identical to the last line shown are not output, but are indicated with an '*' in column 1.

**w**[*n*] Specifies the number of input bytes to be interpreted and displayed on each output line. If **w** is not specified, 16 bytes are read for each display line. If *n* is not specified, it defaults to 32.

**x** Interpret (short) words as hexadecimal.

An upper case format character implies the long or double precision form of the object.

The *offset* argument specifies the byte offset into the file where dumping is to commence. By default this argument is interpreted in octal. A different radix can be specified; if '.' is appended to the argument, then *offset* is interpreted in decimal. If *offset* begins with **x** or **0x**, it is interpreted in hexadecimal. If **b (B)** is appended, the offset is interpreted as a block count, where a block is 512 (1024) bytes. If the *filename* argument is omitted, the *offset* argument must be preceded by '+'.

The radix of the displayed address will be the same as the radix of the *offset*, if specified; otherwise it will be octal.

*label* will be interpreted as a pseudo-address for the first byte displayed. It will be shown in () following the file offset. It is intended to be used with core images to indicate the real memory address. The syntax for *label* is identical to that for *offset*.

**SYSTEM V DESCRIPTION**

The **s** format interprets (short) words as signed decimal, rather than searching for strings; the **S** format searches for strings.

**ENVIRONMENT**

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **od**. On entry to **od**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

**SEE ALSO**

**adb**(1), **dbx**(1), **dbxtool**(1), **locale**(5), **iso_8859_1**(7)

**BUGS**

A file name argument cannot start with '+'. A hexadecimal offset cannot be a block count. Only one file name argument can be given.

It is an historical botch to require specification of object, radix, and sign representation in a single character argument.

**NAME**

old-compact, old-uncompact, old-ccat – compress and uncompress files, and cat them

**SYNOPSIS**

**/usr/old/compact** [ *filename*... ]

**uncompact** [ *filename*... ]

**ccat** [ *filename*... ]

**DESCRIPTION**

**Note:** This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**compact** compresses the named files using an adaptive Huffman code. If no file names are given, the standard input is compacted to the standard output. **compact** operates as an on-line algorithm. Each time a byte is read, it is encoded immediately according to the current prefix code. This code is an optimal Huffman code for the set of frequencies seen so far. It is unnecessary to prepend a decoding tree to the compressed file since the encoder and the decoder start in the same state and stay synchronized. Furthermore, **compact** and **uncompact** can operate as filters. In particular:

... | **compact** | **uncompact** | ...

operates as a (very slow) no-op.

When an argument **file** is given, it is compacted and the resulting file is placed in **file.C**; **file** is removed. The first two bytes of the compacted file code the fact that the file is compacted. This code is used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical values of compression are: Text (38%), Pascal Source (43%), C Source (36%) and Binary (19%). These values are the percentages of file bytes reduced.

**uncompact** restores the original file from a file called *file*.C which was compressed by **compact.** If no file names are given, the standard input is uncompacted to the standard output.

**ccat** cats the original file from a file compressed by **compact,** without uncompressing the file.

**FILES**

| | |
|---|---|
| ***.C** | compacted file created by compact, removed by uncompact |

**SEE ALSO**

Gallager, Robert G., *Variations on a Theme of Huffman*, *I.E.E.E. Transactions on Information Theory*, vol. IT-24, no. 6, November 1978, pp. 668 - 674.

**NAME**

old-eyacc – modified yacc allowing much improved error recovery

**SYNOPSIS**

**/usr/old/eyacc** [ –v ] [ *grammar* ]

**DESCRIPTION**

**eyacc** is a version of **yacc**(1), that produces tables used by the Pascal system and its error recovery routines. **eyacc** fully enumerates test actions in its parser when an error token is in the look-ahead set. This prevents the parser from making undesirable reductions when an error occurs before the error is detected. The table format is different in **eyacc** than it was in the old **yacc**, as minor changes had been made for efficiency reasons.

**SEE ALSO**

**yacc**(1)

*Practical LR Error Recovery* by Susan L. Graham, Charles B. Haley and W. N. Joy; SIGPLAN Conference on Compiler Construction, August 1979.

**BUGS**

**pc** and its error recovery routines should be made into a library of routines for the new **yacc**.

## NAME

old-filemerge – window-based file comparison and merging program

## SYNOPSIS

**/usr/old/filemerge** [ **–br** ] [ **–a** *ancestor* ] [ **–l** *listfile* ] [ *leftfile* [ *rightfile* [ *outfile* ] ] ]

## DESCRIPTION

Note: This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**filemerge** is a window-based version of **diff**(1), for comparing and merging text files. It displays two files for side-by-side comparison, each in a read-only text-subwindow. Beneath them, an editing subwindow can be used to construct a *merged* version—one which contains selected lines from either or both input files, along with any additional edits you may make.

*leftfile* and *rightfile* are the files to be compared, and *outfile* is name of the file containing the merged version. If *outfile* is a directory, then the output is placed in the file *outfile/leftfile*. If *outfile* is omitted, the output file is named **filemerge.out** by default. If no filename arguments are given, you can enter them from within the tool itself.

## OPTIONS

**–b** Ignore leading blanks in comparisons.

**–r** Read-only mode. Do not display the editing subwindow.

**–a** *ancestor*

Compare both files with respect to *ancestor*. A minus-sign indicates lines that have been deleted relative to the ancestor. A plus-sign indicates lines added relative to the ancestor.

**–l** *listfile*

Process a list of filename pairs. With this option, *leftfile* and *rightfile* are the names of directories, and *listfile* contains a list of filenames that appear in both. **filemerge** compares the versions of each file between the two directories, and allows you to create a merged version (typically in the directory *outfile*). The SHIFT-**Load** command button, which is selected by holding the SHIFT key while clicking on the **Load** button, reads in the next pair named in the list. If *listfile* is '-', then the list of files is read from the standard input.

## USAGE

The text in the editing subwindow (*outfile*) is initially the same as that in *leftfile*. To construct a merged version, you can directly edit the text of *outfile* with textedit commands, or you can change a selected difference to match *rightfile* (the one on the right) by clicking the **Right** button in the top panel.

### Differences

At any given time, one of the displayed "differences" is *current*. The current difference is indicated by emboldening the symbol adjacent to each line, and also by the notation "*i* **of** *n*" displayed in the control panel. Once a difference is current, you can use the **Left** and **Right** buttons to apply either the left-hand or the right-hand version of the text to *outfile*. The **Next** and **Prev** buttons select the next or previous difference, respectively.

### Property Sheet

You can customize **filemerge** using the property sheet to set or alter various display and control options. To bring up the property sheet, press the **Props** function key (typically L3) while the mouse is over any part of the **filemerge** window.

### Menus

There are pop-up menus associated with several of the control panel items, and a menu associated with the editing subwindow. The former provide to select any command function obtained with a modified mouse-button (such as SHIFT-**Next** ); the editing subwindow's menu has items that control the filename and directory location of the merged output. To bring up a menu, move the mouse-cursor to the command button, or to the editing subwindow, and hold down the RIGHT mouse-button. Select a desired menu item by releasing the mouse-button after moving the cursor on top of it.

**Command Buttons**

**Next** Make the next difference current. The subwindow scrolls, if necessary, to display it.

SHIFT-**Next** 12 Make the first difference current. (Also a menu item from the **Next** menu.)

**Prev** Make the previous difference current.

SHIFT-**Prev** 12 Make the last difference current. (Also a menu item from the **Prev** menu.)

**Right** Apply right-hand version of the current difference to *outfile*. If **autoadvance** is in effect, advance to the next difference.

SHIFT-**Right** 12 Apply the right-hand version and advance to the next difference, unless **autoadvance** is in effect. (Also a menu item from the **Right** menu.)

CTRL-**Right** Apply the right-hand version for the current difference, and for all subsequent differences up to the end of the file.

**Left** Apply the left-hand version of the current difference.

**Undo** Undo the last **Right** or **Left** operation. You can **Undo** up to 100 stacked operations. You cannot undo an **Undo**.

SHIFT-**Undo** 12 Undo all the operations since the last **Load**, or the last 100 operations.

**Scroll-Lock** When in effect, the three text-subwindows scroll in unison. Otherwise each subwindow scrolls independently.

*i* **of** *n* The number of the current difference, *i*, out of *n* detected differences. Popping up a menu on this item allows you to jump to a selected difference.

**Load** Load the files whose names appear by the prompts **File1:** and **File2:**.

SHIFT-**Load** 12 When the **–l** option is used, load the files from the directories shown in **File1** and **File2** corresponding to the next name in the list (taken from the *listfile* argument).

**Done** Save *outfile* and close the tool. The name used to save the file appears in the namestripe, in the same fashion as **textedit**(1).

SHIFT-**Done** 12 Save without closing. You can also save the merged version using the **Save** item in the editing subwindow's menu.

**Quit** Exit the tool. You must explicitly save your merged *outfile*, either with the **Done** button or the **Save** item in the editing subwindow's menu.

**Properties**

Hitting the L3 function key brings up a property sheet that controls several **filemerge** parameters. The information in the property sheet is stored in the file **~/.filemergerc**. The property panel items have the following meanings:

**Apply** Any changes you have made to the property sheet will now take effect.

**Reset** Reset the property sheet to the state it had at the time of the last **Apply**.

**Done** Close the property sheet.

**autoadvance** Advance to the next difference after each **Left** or **Right**operation.

**Toplines** Number of lines in the top two subwindows.

**Bottomlines** Number of lines in the bottom subwindow.

**Columns** Number of columns in the left (and also right) subwindow.

**FILES**

**~/.filemergerc** file storing property sheet information
**filemerge.out** default output file

**SEE ALSO**

**diff**(1), **sdiff**(1V), **textedit**(1)

**BUGS**

Using the **Find** function key gets the subwindows out of sync for scrolling. To resync them, turn **Scroll-Lock** first off, and then on.

## NAME

old-make – maintain, update, and regenerate groups of programs

## SYNOPSIS

**/usr/old/make** [ **–f** *makefile* ] ... [ **–bdeikmnpqrsSt** ] [ *target* ... ] [ *macro-name=value* ... ]

## DESCRIPTION

**make** executes a list of shell commands associated with each *target*, typically to create or update a file of the same name. *makefile* contains entries for targets that describe how to bring them up to date with respect to the files and/or other targets on which each depends, called *dependencies*.

A target is out of date when the file it describes is missing, or when one (or more) of its dependency files has a more recent modification time than that of the target file. **make** recursively scans the list of dependencies for each *target* argument (or the first *target* entry in the *makefile* if no *target* argument is supplied) to generate a list of targets to check. It then checks, from the bottom up, each target against any files it depends on to see if it is out of date. If so, **make** rebuilds that target.

To rebuild a target, **make** executes the set of shell commands, called a *rule*, associated with it. This rule may be listed explicitly in a makefile entry for that target, or it may be supplied implicitly by **make**.

If no *makefile* is specified on the command line, **make** uses the first file it finds with a name from the following list:

**makefile, Makefile, s.makefile, s.Makefile, SCCS/s.makefile, SCCS/s.Makefile.**

If no *target* is specified on the command line, **make** uses the first target defined in *makefile*. If a *target* has no *makefile* entry, or if its entry has no rule, **make** attempts to update that target using an implicit rule.

## OPTIONS

**–f** *makefile*
: Use the description file *makefile*. A '–' as the *makefile* argument denotes the standard input. The contents of *makefile*, when present, override the builtin rules. When more than one '**–f** *makefile*' argument pairs appear, **make** takes input from each *makefile* in the order listed (just as if they were run through **cat**(1V)).

**–b**
: This option has no effect, but is present for compatibility reasons.

**–d**
: Debug mode. Print out detailed information on files and times examined.

**–e**
: Environment variables override assignments within makefiles.

**–i**
: Ignore error codes returned by invoked commands.

**–k**
: When a nonzero error status is returned by an invoked command, abandon work on the current target but continue with other branches that do not depend on that target.

**–n**
: No execution mode. Print commands, but do not execute them. Even lines beginning with an @ are printed. However, if a command line contains the $(MAKE) macro, that line is always executed (see the discussion of MAKEFLAGS in **Environment Variables and Macros**).

**–p**
: Print out the complete set of macro definitions and target descriptions.

**–q**
: Question mode. **make** returns a zero or nonzero status code depending on whether or not the target file is up to date.

**–r**
: Do not use the implicit rules **make** supplies by default. Implicit rules defined in the makefile remain in effect.

**–s**
: Silent mode. Do not print command lines before executing them.

**–S**
: Undo the effect of the **–k** option.

**–t**
: Touch the target files (bringing them up to date) rather than performing commands listed in their rules.

*macro-name=value*

Macro definition. This definition overrides any definition for the specified macro that occurs in the makefile itself, or in the environment. See **Macros** and *Environment Variables and Macros*, for details.

## USAGE

Refer to *SunOS User's Guide: Doing More* and *Programming Utilities and Libraries* for tutorial information about **make**.

### Targets and Rules

There need not be an actual file named by a target, but every dependency in the dependency list must be either the name of a file, or the name of another target.

If the target has no dependency list and no rule, or if the target has no entry in the makefile, **make** attempts to produce an entry by selecting a rule from its set of implicit rules. If none of the implicit rules apply, **make** uses the rule specified in the **.DEFAULT** target (if it appears in the makefile). Otherwise **make** stops and produces an error message.

### Makefile Target Entries

A target entry has the following format:

*target* ... : [*dependency*] ... [; *command*] ...
[*command*]
...

The first line contains the name of a target (or a space-separated list of target names), terminated with a colon (:). This may be followed by a *dependency*, or a dependency list that **make** checks in the order listed. The dependency list may be terminated with a semicolon (;), which in turn can be followed by a Bourne shell command. Subsequent lines in the target entry begin with a TAB, and contain Bourne shell commands. These commands comprise a rule for building the target, and are performed when the target is updated by **make**.

Shell commands may be continued across input lines by escaping the NEWLINE with a backslash (\). The continuing line must also start with a TAB.

To rebuild a target, **make** expands any macros, strips off initial TAB characters and passes each resulting command line to a Bourne shell for execution.

The first nonblank line that does not begin with a TAB or # begins another target or macro definition.

### Makefile Special Characters

**::** Conditional dependency branch. When used in place of a colon (:) the double-colons allow a target to be checked and updated with respect to more than one dependency list. The double-colons allow the target to have more than one branch entry in the makefile, each with a different dependency list and a different rule. **make** checks each branch, in order of appearance, to see if the target is outdated with respect to its dependency list. If so, **make** updates the target according to dependencies and rule for that branch.

**#** Start a comment. The comment ends at the next NEWLINE.

**$** Macro expansion. See **Macros**, below, for details.

**–** Following the TAB, if the first character of a command line is a '–', **make** ignores any nonzero error code it may return. **make** normally terminates when a command returns nonzero status, unless the **–i** or **–k** options are in effect.

**@** Following the TAB, if the first character is a '@', **make** does not print the command line before executing it.

If '–' and '@' appear as the first two characters after the TAB, both apply.

**$$** The dollar-sign, escaped from macro expansion. Can be used to pass variable expressions beginning with $ to the shell.

**Command Execution**

Command lines are executed one at a time, *each by its own shell*. Shell commands, notably **cd**, are ineffectual across an unescaped NEWLINE in the makefile. A line is printed (after macro expansion) as it is executed, unless it starts with a '@', there is a **.SILENT** entry in the dependency hierarchy of the current target, or **make** is run with the **–s** option. Although the **–n** option specifies printing without execution, lines containing the macro **$(MAKE)** are executed regardless, and lines containing the **@** special character are printed. The **–t** (touch) option updates the modification date of a file without executing any rules. This can be dangerous when sources are maintained by more than one person.

To take advantage of the Bourne shell **if** control structure for branching, use a command line of the form:

**if**
*expression*; \
**then**
*command*; \
*command*; \
...
**elif**
*expression*; \
...
**else**
*command*; \
**fi**

Although composed of several input lines, the escaped NEWLINE characters insure that **make** treats them all as one command line. To take advantage of the Bourne shell **for** control statement, use a command line of the form:

**for** *var* **in** *list* ; **do** \
*command* ; \
...
**done**

To write shell variables, use double dollar-signs (**$$**). This escapes expansion of the dollar-sign by **make**.

**Signals**

INT and QUIT signals received from the keyboard halt **make** and remove the target file being processed (unless it is in the dependency list for **.PRECIOUS**).

**Special-Function Targets**

When incorporated in a makefile, the following target names perform special functions.

**.DEFAULT** If this target is defined in the makefile, its rule is used when there is no entry in the makefile for a given target and none of the implicit rules applies. **make** ignores the dependency list for this target.

**.PRECIOUS** List of files not to delete. Files listed as dependencies for this target are not removed if **make** is interrupted while rebuilding them.

**.SILENT** Run silently. When this target appears in the makefile, **make** does not echo commands before executing them.

**.IGNORE** Ignore errors. When this target appears in the makefile, **make** ignores nonzero error codes returned from commands.

**.SUFFIXES** The suffixes list for selecting implicit rules (see **Implicit Rules**).

**Include Files**

**make** has an include file capability. If the word **include** appears as the first seven letters of a line, and is followed by a SPACE or a TAB, the string that follows is taken as a filename. The text of the named file is read in at the current location in the makefile. **include** files can be nested to a depth of no more than about 16.

**Macros**

Entries of the form

*macro-name=value*

define macros. *name* is the name of the macro, and *value*, which consists of all characters up to a comment character or unescaped NEWLINE, is the value. Words in a macro value are delimited by SPACE, TAB, and escaped NEWLINE characters, and the terminating NEWLINE.

Subsequent references to the macro, of the forms: $(*name*) or ${*name*} are replaced by *value*. The parentheses or brackets can be omitted in a reference to a macro with a single-character name.

Macros definitions can contain references to other macros, but the nested references aren't expanded immediately. Instead, they are expanded along with references to the macro itself.

Substitutions within macros can be made as follows:

$(*name:str1=str2*)

where *str1* is either a suffix, or a word to be replaced in the macro definition, and *str2* is the replacement suffix or word.

**Dynamically Maintained Macros**

There are several dynamically maintained macros that are useful as abbreviations within rules.

**$*** The basename of the current target. It is assigned only for implicit rules.

**$<** The name of the file on which the target is assumed to depend. This macro is only assigned for implicit rules, or within the **.DEFAULT** target's rule.

**$@** The name of the current target. It is assigned only for rules in targets that are explicitly defined in the makefile.

**$?** The list of dependencies with respect to which the target is out of date. This macro is assigned only for explicit rules.

**$%** The library member. The **$%** macro is only evaluated when the target is an archive library member of the form: *lib*(*file*.**o** ). In this case, **$@** evaluates to *lib* and **$%** evaluates to the library member, *file*.**o**.

All of these macros but **$?** can be modified to apply either to the filename part, or the directory part of the strings they stand for, by adding an upper case **F** or **D**, respectively (and enclosing the resulting name in parentheses or braces). Thus, '**$(@D)**' refers to the directory part of the string '**$@**'. If there is no directory part, '**.**' is generated.

**Environment Variables and Macros**

After reading in its implicit rules, **make** reads in variables from the environment, treating them as if they were macro definitions. Only then does **make** read in a makefile. Thus, macro assignments within a makefile override environment variables, provided that the **–e** option is not in effect. In turn, **make** exports environment variables to each shell it invokes. Macros not read in from the environment are *not* exported.

The **MAKEFLAGS** macro is a special case. When present as an environment variable, **make** takes its options (except for **–f**, **–p**, and **–d**) from **MAKEFLAGS** in combination with any flags entered on the command line. **make** retains this combined value, exports it automatically to each shell it forks, and reads its value to obtain options for any **make** commands it invokes. Note, however that flags passed with **MAKEFLAGS** even though they are in effect, are not shown in the output produced by **make**.

The **MAKE** macro is another special case. It has the value **make** by default, and temporarily overrides the **–n** option for any line that contains a reference to it. This allows nested invocations of **make** written as:

**$(MAKE)** ...

to run recursively, so that the command **make –n** can be used to test an entire hierarchy of makefiles.

For compatibility with the 4.2 BSD **make**, the **MFLAGS** macro is set from the **MAKEFLAGS** variable by prepending a '–'. **MFLAGS** is not exported automatically.

**make** supplies the following macros for compilers and their options:

| | | | |
|---|---|---|---|
| **CC** | **C** compiler, **cc** (1V) | **CFLAGS** | **C** compiler options |
| **FC** | FORTRAN 77 compiler, **f77** (1) | **FFLAGS** | FORTRAN 77 compiler options |
| | | **RFLAGS** | FORTRAN 77 compiler options with Ratfor (**.r**) source files |
| **PC** | Pascal compiler, **pc** (1) | **PFLAGS** | Pascal compiler options |
| **M2C** | Modula-2 compiler | **M2FLAGS** | Modula-2 compiler options |
| **GET** | **sccs** (1) **get** command | **GFLAGS** | **sccs get** options |
| **AS** | the assembler, **as** (1) | **ASFLAGS** | assembler options |
| **LD** | the linker, **ld** (1) | **LDFLAGS** | linker options |
| **LEX** | **lex** (1) | **LFLAGS** | **lex** options |
| **YACC** | **yacc** (1) | **YFLAGS** | **yacc** options |

Unless these macros are read in as environment variables, their values are not exported by **make**. If you run **make** with any these set in the environment, it is a good idea to add commentary to the makefile to indicate what value each takes. If **–r** is in effect, **make** ignores these macro definitions.

When set to a single-word value such as **/usr/bin/csh**, the **SHELL** macro indicates the name of an alternate shell to use for invoking commands. Note: to improve normal performance **make** executes command lines that contain no shell metacharacters directly. Such builtin commands as **dirs**, or **set** in the C shell are not recognized unless the command line includes a metacharacter (for instance, a semicolon).

**Implicit Rules**

**make** supplies implicit rules for certain types of targets that have no explicit rule defined in the makefile. For these types of targets, **make** attempts to select an implicit rule by looking for an association between the target and a file in the directory that shares its basename. That file, if found, is presumed to be a dependency file. The implicit rule is selected according to the target's suffix (which may be null), and that of the dependency file. If there is no such dependency file, if the suffix of either dependency or target is not the suffixes list, or if there is no implicit rule defined for that pair of suffixes, no rule is selected. **make** either uses the default rule that you have supplied (if any), or stops.

The suffixes list is a target with each known suffix listed as a dependency, by default:

**.SUFFIXES: .o .c .c~ .mod .mod~ .sym .def .def~ .p .p~ .f .f~ .r .r~ .y .y~ .l .l~**
**.s .s~ .sh .sh~ .h .h~**

Multiple suffix-list targets accumulate; a **.SUFFIXES** target with no dependencies clears the list of suffixes. Order is significant; **make** selects a rule that corresponds to the target's suffix and the first dependency-file suffix found in the list.

A tilde (~) refers to the **s.***prefix* of an SCCS history file (see **sccs**(1)). If **make** cannot locate a history file (with a name of the form **s.***basename.suffix*) in the current working directory, it checks for one in the SCCS subdirectory (if that directory exists) for one from which to **sccs-get**(1) the dependency file.

An implicit rule is a target of the form:

*dt*:
    *rule*

where *t* is the suffix of the target, *d* is the suffix of the dependency, and *rule* is the implicit rule for building such a target from such a dependency. Both *d* and *t* must appear in the suffixes list for **make** to recognize the target as one that defines an implicit rule.

An implicit rule with only one suffix describes how to build a target having a null (or no) suffix, from a dependency having the indicated suffix. For instance, the **.c** rule describes how to build the executable *file* from a **C** source file, *file*.**c**.

Implicit rules are supplied for the following suffixes and suffix pairs:

**.c .c~ .p .p~ .mod .mod~ .f .f~ .F .F~ .r .r~ .sh .sh~ .c.o .c~.o .c~.c .p.o .p~.o .p~.p .mod.o .mod~.o .mod~.mod .def.sym .def~.sym .def~.def .f.o .f~.f .F.o .F~.o .F~.F .r.o .r~.o .r~.r .s.o .s~.o .s~.s .sh~.sh .y.o .y~.o .l.o .l~.o .y.c .y~.c .y~.y .l.c .l~.c .l~.l .c.a .c~.a .s~.a .h~.h**

These rules can be changed within a makefile, and additional implicit rules can be added. To print out **make**'s internal rules, use the following command. Note: this command only works with the Bourne Shell:

**$ make –fp – 2>/dev/null </dev/null**

If you are using the C shell, use this command to print out **make**'s internal rules:

**example% (make –fp – </dev/null >/dev/tty) >&/dev/null**

**Library Maintenance**

If a target name contains parentheses, as with:

**lib.a(member)**

it is assumed to be the name of an archive (**ar**(1V)) library. The string within the parentheses refers to a member of the library. (If the string contains more than one word, the only first word is used.) A member of an archive can be explicitly made to depend on a file with a matching filename. For instance, given a directory that contains the files **mem1.c** and **mem2.c**, along with a makefile with the entries:

```
lib.a: lib.a(mem1.o) lib.a(mem2.0)

lib.a(mem1.o): mem1.o
        ar rv lib.a mem1.o

lib.a(mem2.o): mem2.o
        ar rv lib.a mem2.o
```

**make**, when run, compiles the .c files into relocatable object (**.o**) files using the **.c.o** implicit rule. It then loads the freshly compiled version of each file into the library according to the explicit rules in the **lib.a**()targets.

Implicit rules pertaining to archive libraries have the form *.XX*.**a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires that *XX* to be different from the suffix of the archive member itself. For instance, the target **lib**(*file*.**o**) cannot depend upon the *file*.**o** explicitly, but instead, must be made to depend on a source file, such as *file*.**c**. For this reason it is recommended that you define an explicit target in the makefile for each library member to maintain, as shown above.

A target name of the form

*library*((*entry-point*))

refers to the member of a randomized object library (see **ranlib**(1)) that defines the symbol *entry-point*.

**EXAMPLES**

This *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) along with a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o –o $@

a.o: incl.h a.c
        cc –c a.c

b.o: incl.h b.c
        cc –c b.c
```

The following *makefile* uses the builtin inference rules to express the same dependencies:

```
pgm: a.o b.o
        cc a.o b.o -o pgm

a.o b.o: incl.h
```

**FILES**

*[Mm]akefile*
**s.***[Mm]akefile*
**SCCS/s.***[mM]akefile*

**DIAGNOSTICS**

**Don't know how to make** *target* **. Stop.**
There is no makefile entry for *target*, and none of **make**'s implicit rules apply (there is no dependency file with a suffix in the suffixes list, or the target's suffix is not in the list).

*** *target* **removed.**
**make** was interrupted in the middle of trying to build *target*. Rather than leaving a partially-completed version that is newer than its dependencies, make removes the file associated with *target*.

*** **Error code** *n*.
The previous shell command returned a nonzero error code. In this case *make* stops, unless either the **–k** or the **–i** option is set, the target **.IGNORE** appears, or the command is prefixed with a '–' in the makefile.

*** *signal message*
The previous shell command was aborted due to a signal. If '**– core dumped**' appears after the message, a **core** file was created.

**SEE ALSO**

**ar**(1V), **cat**(1V), **cc**(1V), **cd**(1), **csh**(1), **lex**(1), **ranlib**(1), **sccs**(1), **sccs-get**(1), **sh**(1)

*SunOS User's Guide: Doing More*
*Programming Utilities and Libraries*

**BUGS**

Some commands return nonzero status inappropriately; use **–i** to overcome the difficulty.

Filenames with the characters =, :, and @ will not work.

You cannot build **lib(***file***.o)** from *file***.o**.

The macro substitution **$(a:.o=.c~)** does not work.

Options supplied by **MAKEFLAGS** should appear in output from **make**.

**NAME**

old-prmail – display waiting mail

**SYNOPSIS**

**/usr/old/prmail** [ *user* ] ...

**DESCRIPTION**

Note: This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**prmail** displays waiting mail for you, or the specified *user*s. The mail is not disturbed.

**FILES**

**/var/spool/mail/*** waiting mail files

**SEE ALSO**

**biff**(1), **bin-mail**(1), **from**(1), **mail**(1)

**NAME**

old-pti – phototypesetter interpreter

**SYNOPSIS**

**/usr/old/pti** [ *filename* ... ]

**DESCRIPTION**

Note: This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**pti** shows the commands in a stream from the standard output of **troff**(1) using **troff**'s **–t** option, interpreting the commands as they would act on the typesetter. Horizontal motions show as counts in internal units and are marked with < and > indicating left and right motion. Vertical space is called *leading* and is also indicated.

The output is really cryptic unless you are an experienced C/A/T hardware person. It is better to use '**troff –a**'.

**SEE ALSO**

**troff**(1)

**NAME**

old-setkeys – modify interpretation of the keyboard

**SYNOPSIS**

**/usr/old/setkeys [ reset | nosunview | [ [ lefty ] [ noarrows ] ] ] [ sun1 | sun2 | sun3 ]**

**Sun386i SYNOPSIS**

**setkeys [ reset | nosunview | [ [ lefty ] [ noarrows ] ] ] [ sun1 | sun2 | sun3| sun4 ]**

**DESCRIPTION**

**setkeys** has been superseded by the **Input** category in **defaultsedit**(1), and by the program **input_from_defaults** (1). It is retained for backwards compatibility on Sun-2, Sun-3 and Sun-4 systems.

**Sun386i DESCRIPTION**

**setkeys** changes the kernel's keyboard translation tables, converting a keyboard to one of a number of commonly desired configurations. It takes an indication of the modifications to be performed, and optionally, the kind of keyboard attached to the user's machine. It affects all keyboard input for the machine it is run on (in or out of the window system) until that effect is superseded by rebooting, or by running '**setkeys reset**'.

**OPTIONS**

*modifications*

Empty, or one of **reset**, **nosunview**, or some combination of **lefty** and **noarrows**. By default, the keyboard is set to produce the SunView function-key codes (**Stop**, **Props**, **Front**, **Close**, **Find**, **Again**, **Undo**, **Copy**, **Paste** and **Cut**; *SunView User's Guide*. On Sun-2 and Sun-3 keyboards, this is meaningless; on the Sun-1, those functions are assigned to two columns of the right numeric-function pad.

**lefty** Indicate the SunView functions are to be produced from keys on the right side of the keyboard, convenient for using the mouse in the left hand.

On the Sun-2 and Sun-3 systems, the SunView functions are reflected to the outside columns of the right function pad; those right-side functions are distributed in a more complicated fashion dictated by keeping the arrow keys together; see below. Also, the Line Feed key, immediately below Return, is converted to a second Control.

On the Sun-1, **lefty** is the same as the default, since there is no left function pad.

**noarrows**

Reassign the keys with cursor arrows on their caps to produce simple function codes (so they may be used with filters in the textsw, or mapped input in the ttysw).

**nosunview**

Only valid on Sun-2 and Sun-3 keyboards, or on a Sun-4 keyboard used with 386i architecture. **nosunview** is incompatible with **lefty**, **noarrows**, or **reset**. This option assigns new codes to keys F1 and L2 - L10, codes that are not normally produced anywhere on the keyboard. These codes may be selected by a *mapi* or *mapo* operation defined in a user's **.ttyswrc** file.

This option supports a measure of backwards compatibility to programs that apply some other interpretation to the affected function keys. It allows them to access the new codes when the standard codes would be preempted by SunView functions (for instance, in a **tty**(1) subwindow).

**reset** Incompatible with **lefty**, **noarrows**, or **nosunview**; it causes the keyboard to be reset to its original interpretation.

*keyboard-type*

One of **sun1**, **sun2**, or **sun3**. **sun4** is available with 386i architecture only. Normally, this option is omitted; the type of keyboard attached to the system is obtained from the kernel. If included, the option is believed in preference to the kernel's information. **setkeys** treats Sun-2 and Sun-3 keyboards identically except when the modification is **reset**.

Note: The keyboard type is not necessarily the same as the machine type. A Sun-1 keyboard is the VT100-style keyboard shipped with Model 100Us, while Sun-2 and Sun-3 keyboards may be attached interchangeably to Sun-2 and Sun-3 machines. A Sun-3 keyboard is distinguished by its aerodynamic housing, and the presence of Caps and Alternate keys.

Options may appear in any order, and case is not significant. The accompanying diagrams show the exact distribution of codes for each combination of keyboard and arguments to setkeys.

## EXAMPLES

The command

**setkeys lefty noarrows**

puts the SunView functions on the right pad of the keyboard, replacing arrow keys by the corresponding right-function codes, and displacing right-function codes to the left pad.

The command:

**setkeys sun1 reset**

restores a Sun-1 keyboard to its original arrangement.

## SEE ALSO

**defaultsedit**(1), **input_from_defaults**(1), **kb**(4M)

*SunView User's Guide*

## BUGS

**setkeys** affects the kernel's key tables, which in turn affects all users logged in to the system.

## DIAGRAMS

**Sun-1, reset:**

```
          ^  V  <  >
[       standard        ]     TF1    TF2    TF3    TF4
[        typing         ]     7      8      9      -
[         array         ]     4      5      6      ,
[         ....          ]     1      2      3      En-
                                0             .      ter
```

**default / lefty:**

```
          ^  V  <  >
[       standard        ]     Again  RF1    Stop   RF2
[        typing         ]     Undo   RF3    Props  RF4
[         array         ]     Put    RF5    Front  RF6
[         ....          ]     Get    RF7    Close  RF8
                                Delete Find
```

**default / lefty, noarrow:**

```
          TF1 TF2 TF3 TF4
[       standard        ]     Again  RF1    Stop   RF2
[        typing         ]     Undo   RF3    Props  RF4
[         array         ]     Put    RF5    Front  RF6
[         ....          ]     Get    RF7    Close  RF8
                                Delete Find
```

**Sun-2 & Sun-3,**

**reset / default:**

| | | | TF1 TF2 ... | ] | | | |
|---|---|---|---|---|---|---|---|
| Stop | Again | [ | standard | ] | RF1 | RF2 | RF3 |
| Props | Undo | [ | typing | ] | RF4 | RF5 | RF6 |
| Front | Put | [ | array | ] | RF7 | ^ | RF9 |
| Close | Get | [ | | Retn | < | RF11 | > |
| Find | Delete | [ | | LF | RF13 | V | RF15 |

**noarrows (only):**

| | | | TF1 TF2 ... | ] | | | |
|---|---|---|---|---|---|---|---|
| Stop | Again | [ | standard | ] | RF1 | RF2 | RF3 |
| Props | Undo | [ | typing | ] | RF4 | RF5 | RF6 |
| Front | Put | [ | array | ] | RF7 | RF8 | RF9 |
| Close | Get | [ | | Retn | RF10 | RF11 | RF12 |
| Find | Delete | [ | | LF | RF13 | RF14 | RF15 |

**lefty:**

| | | | TF1 TF2 ... | ] | | | |
|---|---|---|---|---|---|---|---|
| Stop | RF1 | [ | standard | ] | Again | < | Stop |
| RF6 | RF4 | [ | typing | ] | Undo | > | Props |
| RF9 | RF7 | [ | array | ] | Put | ^ | Front |
| RF12 | RF10 | [ | | Retn | Get | RF11 | Close |
| RF15 | RF13 | [ | | Ctrl | Delete | V | Find |

**lefty, noarrows**

| | | | TF1 TF2 ... | ] | | | |
|---|---|---|---|---|---|---|---|
| Stop | RF1 | [ | standard | ] | Again | RF2 | Stop |
| RF6 | RF4 | [ | typing | ] | Undo | RF5 | Props |
| RF9 | RF7 | [ | array | ] | Put | RF8 | Front |
| RF12 | RF10 | [ | | Ret | Get | RF11 | Close |
| RF15 | RF13 | [ | | Ctrl | Delete | RF14 | Find |

**nosunview:**

| | | | LF11 TF2 ... | ] | | | |
|---|---|---|---|---|---|---|---|
| Stop | TF11 | [ | standard | ] | RF1 | RF2 | RF3 |
| LF12 | TF12 | [ | typing | ] | RF4 | RF5 | RF6 |
| LF13 | TF13 | [ | array | ] | RF7 | ^ | RF9 |
| LF14 | TF14 | [ | | Ret | < | RF11 | > |
| LF15 | TF15 | [ | | LF | RF13 | V | RF15 |

## NAME

old-sun3cvt – convert large Sun-2 system executables to Sun-3 system executables

## SYNOPSIS

**/usr/old/sun3cvt** [ *oldfile* [ *newfile* ] ]

## DESCRIPTION

Note: This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**sun3cvt** converts an old Sun-2 system program file (predating Sun release 3.0) into a Sun-3 system executable file.

The default *oldfile* is **a.out**. The default *newfile* is **sun3.out**.

**sun3cvt** attempts to create a file of the same type (magic number). However, for sharable-text files with less than 128kb of text, the new file will not be sharable (since Sun-3 data segments must begin on 128kb boundaries). Also, most programs have some text in the data segment as a consequence of the new larger Sun-3 system page and segment sizes.

**execve**(2V) executes an old Sun-2 system program file, but the program's text is not sharable. Old pure-text programs with text segments larger than 128kb can be made sharable on machines running release 3.0 or subsequent releases by using **sun3cvt**.

## FILES

| | |
|---|---|
| **a.out** | default *oldfile* |
| **sun3.out** | default *newfile* |

## SEE ALSO

**execve**(2V)

**NAME**

old-syslog – make a system log entry

**SYNOPSIS**

**/usr/old/syslog** [ – ] [ **–p** ] [ **–i** *name* ] [ *–level* ] [ message... ]

**DESCRIPTION**

Note: this program is considered to be obsolete, and will not be distributed or supported in future Sun releases. See **logger**(1) to add system log entries.

**syslog** sends the specified message (or the standard input if '–' is specified) as a system log entry to the syslog daemon. The log entry is sent to the daemon on the machine specified by the *loghost* entry in the **/etc/hosts** file.

**OPTIONS**

– Each line of the standard input is sent as a log entry.

**–p** **syslog** logs its process ID in addition to the other information.

**–i** *name* Specify the name to be used as the "ident" for the log entry.

*–level* The message will be logged at the specified level. The level can be specified numerically, in the range 1 through 9, or symbolically using the names specified in the include file **<syslog.h>** (with the leading LOG_ stripped off). **syslog-HELP** lists the valid symbolic level names. Only the super-user can make log entries at levels less than or equal to **SALERT**.

**FILES**

**/usr/etc/syslogd** syslog daemon

**SEE ALSO**

**logger**(1), **syslog**(3), **syslogd**(8)

**NAME**

old-vc – version control

**SYNOPSIS**

**/usr/old/vc** [ **–a** ] [ **–s** ] [ **–t** ] [ **–cc** ] [ *keyword=value*]...

**DESCRIPTION**

The **vc** command copies lines from the standard input to the standard output under control of its *arguments* and the *control statements* encountered in the standard input. In the process of performing the copy operation, a reference to a user-declared *keyword* is replaced by its character-string *value*, when appearing in a plain text line or control statement.

Copying lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified either in control statements, or as **vc** command-line arguments.

**OPTIONS**

**–a** Force replacement of keyword references in *all* text lines, and not just in **vc** statements.

**–s** Silent. Suppress warning messages (but not error messages) that are normally printed on the diagnostic output.

**–t** If a TAB character appears on a line, all characters from the beginning of a line, up to and including the first TAB, are ignored for the purpose of detecting a control statement. If the TAB precedes a control statement, the leading text is discarded.

**–c***c* Specify an alternate control character to use instead of '**:**'.

**USAGE**

**vc** distinguishes between text input lines and version control statement lines. A version control statement (control statement) is a single line beginning with a control character. The default control character is colon (:), except as modified by the **–c***c* option. Input lines beginning with a backslash (\), and followed by a control character, are not control lines and are copied to the standard output as text with the backslash removed. Lines beginning with a backslash, but not followed by a control character, are copied in their entirety.

**Keyword Replacement**

A keyword is composed of 9 or less alphanumeric characters; the first must be alphabetic. A value is any printable ASCII character or character string. An unsigned string of digits is treated as a numeric value in control operations. Keyword values may not contain any SPACE or TAB characters.

Keyword replacement is performed whenever a keyword, surrounded by control characters, is encountered on a version control statement. The **–a** option forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with '\'. If a literal '\' is desired, then it too must be preceded by a '\'.

**Version Control Statements**

**: dcl** *keyword*[,*keyword*]

Declare a keyword. All keywords must be declared.

**: asg** *keyword=value*

Assign a value to a keyword. An **asg** statement overrides any previous assignment for the corresponding keyword, including those on the **vc** command line. Keywords declared, but not assigned values, have null values.

**:if [not]** *condition*
...
**:end** Skip lines of the standard input. When *condition* is TRUE , all lines between the **if** statement and the matching **end** statement are *copied* to the standard output. Otherwise, the intervening lines are discarded and ignored, *including* intervening control statements. Intervening **if** and **end** control statements are recognized solely for the purpose of maintaining the proper **if**–**end** matching. The **not** argument inverts the sense of the condition. When it is used, intervening lines are included in the output only when the conditions is false.

*condition* is a logical expression composed of *comparisons* and logical operators. A *comparison* consists of two text values (may be keyword references) separated by a comparison operator. Each value must be separated from all operators by at least one SPACE . Numeric strings are treated as unsigned integers for certain comparisons. The comparison operators are:

| | |
|---|---|
| = | equal (string) |
| != | not equal (string) |
| > | greater than (numeric) |
| < | less than (numeric) |

For instance, the line:

**:if** *xxx* != *yyy*

tests to see whether the string '*xxx*' is not equal to '*yyy*', which is true; subsequent intervening lines are therefore included.

The logical sense of comparisons can be combined using the logical operators (in order of precedence):

| | |
|---|---|
| ( ) | logical grouping |
| & | and |
| \| | or |

For instance, the line

**:if** *xxx* = *yyy* | *xxx* != *yyy*

is true because either comparison will make it true, while

**:if** *xxx* = *yyy* & *xxx* != *yyy*

is false, because in this case, both comparisons must be true.

**::***text* Force keyword replacement on lines that are copied to the standard output, independent from the **–a** option. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file.

**:on**
**:off** Turn on or off keyword replacement on all lines.

**:ctl** *c* Change the control character to *c*.

**:msg** *message*
Print *message* on the diagnostic output.

**:err** *message*
Print the given message, followed by:

**ERROR: err statement on line ... (915)**

on the diagnostic output; **vc** halts execution, and returns an exit code of **1**.

**SEE ALSO**
**sccs-help**(1)

**DIAGNOSTICS**

Use **sccs-help**(1) for explanations.

## NAME

on – execute a command on a remote system, but with the local environment

## SYNOPSIS

**on** [ **–i** ] [ **–d** ] [ **–n** ] *host command* [ *argument* ] ...

## DESCRIPTION

The **on** program is used to execute commands on another system, in an environment similar to that invoking the program. All environment variables are passed, and the current working directory is preserved. To preserve the working directory, the working file system must be either already mounted on the host or be exported to it. Relative path names will only work if they are within the current file system; absolute path names may cause problems.

The standard input is connected to the standard input of the remote command, and the standard output and the standard error from the remote command are sent to the corresponding files for the **on** command.

## OPTIONS

**–i** Interactive mode. Use remote echoing and special character processing. This option is needed for programs that expect to be talking to a terminal. All terminal modes and window size changes are propagated.

**–d** Debug mode. Print out some messages as work is being done.

**–n** No Input. This option causes the remote program to get EOF when it reads from the standard input, instead of passing the standard input from the standard input of the **on** program. For example, **–n** is necessary when running commands in the background with job control.

## SEE ALSO

**chkey**(1), **publickey**(5), **exports**(5), **rexd**(8C)

## DIAGNOSTICS

**unknown host** Host name not found.

**cannot connect to server**
Host down or not running the server.

**can't find** Problem finding the working directory.

**can't locate mount point**
Problem finding current file system. **RPC: Authentication error** The server requires des authentication and you do not have a secret key registered with keyserv. Perhaps you logged in without a password. Try to keylogin. If that fails try to set your publickey with chkey.

Other error messages may be passed back from the server.

## BUGS

The SunView window system can get confused by the environment variables.

When the working directory is remote mounted over NFS, a ^**Z** hangs the window.

Root cannot use **on**.

## NAME

organizer – file and directory manager

## SYNOPSIS

**organizer** [ **–A** *time* ]

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

**organizer** is a SunView application for viewing and manipulating files and directories. It performs many of the functions of the **ls**(1V), **cd**(1), **chmod**(1V), **cp**(1), **mv**(1), **mkdir**(1), **rmdir**(see **rm**(1)), and **dump**(8) commands, and with a visual interface.

At any given time, the **organizer** window normally shows the files and directories in a single directory, representing each file or directory with an appropriate illustrated icon. The illustration indicates whether a file is a directory, contains text, is an executable program, or optionally a user-defined file type.

When **organizer** is switched into Map mode, the icons are arranged to indicate the hierarchy of files and directories. Double clicking on a directory icon shows the contents of that directory in a new column.

Several display modes are available, and can be set for an individual **organizer** window or for all **organizer** windows. You can select whether hidden files are shown, whether just the name, the name and information, or name and icon are shown for each file and directory, and how the contents are sorted.

Text files can be "edited" by double clicking on the file's icon. The contents of the file are then shown and can be edited in a separate text editor window. In the **.orgrc** file you can specify the EXECUTE, EDIT, and PRINT applications for your own user-defined file types.

You can move down through the directory hierarchy by double clicking on a directory icon, and up by double clicking on the parent directory name on the ancestor list in the upper left corner of the **organizer** panel.

Copying, moving, and deleting require you to select one or more files. To select a file, click the left button on it (do not double click — this will open the file). To select additional files to be operated on, click the middle button on each additional file.

Copying and moving operations require a destination directory. After the files are selected, change directories to the desired destination as described above, and then "drop" the files with the Drop button on the command panel. If the copy involves overwriting an identically named file, an alert will allow you to confirm that you want to overwrite the file. If you copy a file and then "drop" it in the same directory, **organizer** will prepend **copy_of_** to the file name of the new file.

## OPTIONS

**–A** *time* Sets autoclose for *time* minutes. If **organizer** is not used for *time* minutes, it closes automatically.

## FILES

| | |
|---|---|
| **/usr/include/images/*** | file and directory icons |
| **~/.orgrc** | |
| **../.DeletedOrgFiles** | **organizer** creates this directory to store files so you can undo the last thing you did, for example, deleting files. |

## SEE ALSO

**mkdir**(1), **mv**(1), **rm**(1), **orgrc**(5), **restore**(8)

*Sun386i User's Guide*
*Sun386i Advanced Skills*

**NAME**

overview – run a program from SunView that takes over the screen

**SYNOPSIS**

**overview** [ **–w** ] [ *generic_tool_options* ] *program_name* [ *arguments* ] ...

**AVAILABILITY**

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

Bitmap graphics based programs that are not SunView based can be run from SunView using **overview**. **overview** shows an icon in SunView when **overview** is brought up iconic (**–Wi** generic tool option; see **sunview**(1)) or when the program being run by **overview** is suspended (for example, using CTRL-Z). Opening the **overview** icon, or starting **overview** non-iconic, starts the program named on the command line. **overview** suppresses SunView so that SunView window applications will not interfere with the program's display output or input devices.

**overview** runs programs that fit the following profile:

*own display* — The program needs to own the bits on the screen. It does not use the *sunview* library to arbitrate the use of the display and input devices between processes.

*keyboard input from* **stdin** — The program takes keyboard input from *stdin* directly.

*mouse input from* **/dev/mouse** — The program takes locator input from the mouse directly.

**OPTIONS**

**–w** This option is used to specify that the program being run creates its own SunWindows window in order to receive the serialized input stream from the keyboard and mouse that is provided by the SunWindows kernel driver. **–w** tells **overview** to not convert SunWindows input into ASCII which is then sent to the program being run under **overview** via a pty. X and NeWS are programs that fall in this category (as of Dec 86. This is subject to change in the future.)

**SEE ALSO**

**sunview**(1)

**BUGS**

Users of **overview** on a Sun-3/110 should be aware of the impact of plane groups on pre-3.2 applications. You cannot successfully run pre-3.2 applications under **overview** if **overview** itself is running in the color buffer. If you start **overview** so that it is not running in the overlay plane, then the enable plane is not be properly set up for viewing the application. This means that you cannot run **overview** with the **–Wf** or **–Wb** generic tool arguments. Also, you cannot run **overview** on a desktop created by **sunview** using the **-8bit_color_only** option.

## NAME

pack, pcat, unpack – compress and expand files

## SYNOPSIS

**pack** [ – ] [ **–f** ] *filename*...

**pcat** *filename*...

**unpack** *filename*...

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**pack** attempts to store the specified files in a packed form using Huffman (minimum redundancy) codes on a byte-by-byte basis. Wherever possible (and useful), each input file *filename* is replaced by a packed file *filename*.**z** with the same access modes, access and modified dates, and owner as those of *filename*. If **pack** is successful, *filename* will be removed.

Packed files can be restored to their original form using **unpack** or **pcat**.

The amount of compression obtained depends on the size of the input file and the frequency distribution of its characters.

Because a decoding tree forms the first part of each .**z** file, it is usually not worthwhile to pack files smaller than three blocks unless the distribution of characters is very skewed. This may occur with printer plots or pictures.

Typically, large text-files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression. Their packed versions come in at about 90% of the original size.

No packing will occur if:

the file appears to be already packed
the file name has more than 12 characters
the file has links
the file is a directory
the file cannot be opened
no disk storage blocks will be saved by packing
a file called *name*.**z** already exists
the .**z** file cannot be created
an I/O error occurred during processing

The last segment of the filename must contain no more than 12 characters to allow space for the appended .**z** extension. Directories cannot be packed.

**pcat** does for packed files what **cat**(1V) does for ordinary files, except that **pcat** cannot be used as a filter. The specified files are unpacked and written to the standard output. To view a packed file named *name*.**z** use:

**pcat** *filename*.**z**

or just:

**pcat** *filename*

To make an unpacked copy without destroying the packed version, use

**pcat** *filename* > *newname*

Failure may occur if:

the filename (exclusive of the .z) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of **pack**.

**unpack** expands files created by **pack**. For each file *filename* specified in the command, a search is made for a file called *filename*.z (or just *filename*, if *filename* ends in .z). If this file appears to be a packed, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file. Failure may occur for the same reasons that it may in **pcat**, as well as for the following:

- a file with the "unpacked" name already exists
- the unpacked file cannot be created.

**OPTIONS**

**–** Print compression statistics for the following filename or names on the standard output. Subsequent '–'s between filenames toggle statistics off and on.

**–f** Force packing of *filename*. This is useful for causing an entire directory to be packed, even if some of the files will not benefit.

**DIAGNOSTICS**

**pack** returns the number of files that it failed to compress.

**pcat** returns the number of files it was unable to unpack.

**unpack** returns the number of files it was unable to unpack.

**SEE ALSO**

**cat**(1V)

**NAME**

pagesize – display the size of a page of memory

**SYNOPSIS**

**pagesize**

**DESCRIPTION**

**pagesize** prints the size of a page of memory in bytes, as returned by **getpagesize**(2). This program is useful in constructing portable shell scripts.

**SEE ALSO**

**getpagesize**(2)

## NAME

passwd, chfn, chsh – change local or NIS password information

## SYNOPSIS

**passwd** [ **–l** | **–y** ] [ **–afs** ] [ **–d** [ *username* ] ] [ **–e** *username* ] [ **–F** *filename* ] [ **–n** *numdays username* ] [ **–x** *numdays username* ] [ *username* ]

**chfn** [ **–l** | **–y** ] [ **–f** ] [ **–F** *filename* ] [ *username* ]

**chsh** [ **–l** | **–y** ] [ **–s** ] [ **–F** *filename* ] [ *username* ]

## DESCRIPTION

**passwd** changes (or installs) a password, login shell (**–s** option), or full name (**–f** option) associated with the user *username* (your own by default). **chsh** is equivalent to **passwd** with the **–s** option, and **chfn** is equivalent to **passwd** with the **–f** option.

Use '**passwd –y**' or **yppasswd**(1) to change your password in the Network Information Service (NIS). This will not affect your local password, or your password on any remote machines on which you have accounts. **passwd** calls **yppasswd** automatically if you do not have an entry in the local passwd file, and the **–l** option is not specified.

When changing a password, **passwd** prompts for the old password and then for the new one. You must supply both, and the new password must be typed twice to forestall mistakes.

If password aging is enabled, the first time an ordinary user enters the new password **passwd** checks to see if the old password has "aged" sufficiently. Password "aging" is the amount of time (usually a certain number of days) that must elapse between password changes. If "aging" is insufficient the new password is rejected and **passwd** terminates.

New passwords should be at least five characters long, if they combine upper-case and lower-case letters, or at least six characters long if in monocase. Users that persist in entering shorter passwords are compromising their own security. The number of significant characters in a password is eight, although longer passwords will be accepted.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password. The super-user can change any password and is not forced to comply with password aging requirements.

When changing a login shell, **passwd** displays the current login shell and then prompts for the new one. The new login shell must be one of the approved shells listed in **/etc/shells** unless you are the super-user. If **/etc/shells** does not exist, the only shells that may be specified are **/bin/sh** and **/bin/csh**.

The super-user may change anyone's login shell; normal users may only change their own login shell.

When changing a full name, **passwd** displays the current full name, enclosed between brackets, and prompts for a new full name. If you type a RETURN, the full name is not changed. If the full name is to be made blank, you must type the word "none".

The super-user may change anyone's full name; normal users may only change their own.

## OPTIONS

**–a** Display the name and aging information for all users. Can only be invoked by the super-user.

**–f** Change the full name.

**–l** Change the local password, login shell, or full name. If *username* exists in the local passwd file, this is the default.

**–s** Change the login shell.

**–y** Change passwd, login shell, or full name in the NIS database.

**–d** [*username*]
Display the name and aging information for the caller or the user specified if the invoker has the right privileges.

**-e** *username*
Expire the password for the user name specified. Can only be invoked by the super-user.

**-F** *filename*
Treat *filename* as the password file.

**-n** *numdays username*
Set the maturity time of the password for username. Passwords that have not "aged" enough cannot be changed. Can only be set by the super-user.

**-x** *numdays username*
Set the expiration time of the password for username. Can only be set by the super-user.

**FILES**

| | |
|---|---|
| **/etc/passwd** | file containing all of this information |
| **/etc/shells** | list of approved shells |

**SEE ALSO**

**finger**(1), **login**(1), **yppasswd**(1), **crypt**(3), **passwd**(5)

**NOTES**

Password algorithms do not work with 8-bit characters. This maintains consistency with **login** file naming rules, which do not allow 8-bit characters in **login** names. See **login**(1) for explanations about why **login** is not 8-bit clean.

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

## NAME

paste – join corresponding lines of several files, or subsequent lines of one file

## SYNOPSIS

**paste** *filename1 filename2 ...*
**paste** **–d***list filename1 filename2 ...*
**paste** **–s** [ **–d***list* ] *filename*

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

In the first two forms, **paste** concatenates corresponding lines of the given input files *filename1*, *filename2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). It is the counterpart of **cat**(1V) which concatenates vertically, that is, one file after the other. In the last form above, **paste** replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the TAB character, or with characters from an optionally specified *list*. **paste** can be used as a filter, if – is used in place of a filename.

## OPTIONS

**–d** Without this option, the NEWLINE characters of each but the last file (or last line in case of the **–s** option) are replaced by a TAB character. This option allows replacing the TAB character by one or more alternate characters in *list*. The list is used circularly; when exhausted, it is reused. In parallel merging (no **–s** option), the lines from the last file are always terminated with a NEWLINE character, not from the *list*. *list* may contain the special escape sequences: **\n (NEWLINE), \t (tab), \\ (backslash),** and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell.

**–s** Merge subsequent lines rather than one from each input file. Use TAB for concatenation, unless *list* is specified with **–d**. Regardless of the *list*, the very last character of the file is forced to be a NEWLINE.

## EXAMPLES

List directory in four columns:

```
ls | paste – – – –
```

Combine pairs of lines into lines:

```
paste –s –d"\t\n" filename
```

## SEE ALSO

**cat**(1V), **cut**(1V), **grep**(1V), **pr**(1V)

## DIAGNOSTICS

**line too long**
Output lines are restricted to 511 characters.

**too many files**
Except for **–s** option, no more than 12 input files may be specified.

## NAME

pax – portable archive exchange

## SYNOPSIS

**/usr/5bin/pax** [ **–cimopuvy** ] [ **–f** *archive* [ **–s** *replstr* ] [ **–t** *device* ] [ *pattern* ... ]
**/usr/5bin/pax –r** [ **–cimnopuvy** ] [ **–f** *archive* ] [ **–s** *replstr* ] [ **–t** *device* ] [ *pattern* ... ]
**/usr/5bin/pax –w** [ **–adimuvy** ] [ **–b** *blocking* ] [ **–f** *archive* ] [ **–s** *replstr* ] [ **–t** *device* ] [ **–x** *format* ]
[ *pathname* ... ]
**/usr/5bin/pax –rw** [ **–ilmopuvy** ] [ **–s** *replstr* ] [ *pathname* ... ] *directory*

## AVAILABILITY

**pax** is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**pax** reads and writes archive files which conform to the **Archive/Interchange File Format** specified in *IEEE Std. 1003.1-1988* (POSIX.1). **pax** can also read, but not write, a number of other file formats in addition to those specified in the **Archive/Interchange File Format** description. Support for these traditional file formats, such as V7 **tar**(1) and System V binary **cpio**(1) format archives, is provided for backward compatibility and to maximize portability.

**pax** will also support traditional **cpio** and System V **tar** interfaces if invoked with the name **cpio** or **tar** respectively. See the **paxcpio**(1V) or **ustar**(1V) manual pages for more details.

*directory* specifies the destination directory pathname for copies when both the **–r** and **–w** options are specified. The directory must exist and be writable before the copy or and error results.

*pathname* is a file whose contents are used instead of the files named on the standard input. When a directory is named, all of its files and (recursively) subdirectories are copied as well.

A *pattern* is given in the standard shell pattern matching notation. The default if no *pattern* is specified is '*', which selects all files.

Combinations of the **–r** and **–w** command line arguments specify whether **pax** will read, write or list the contents of the specified archive, or move the specified files to another directory.

If neither the **–r** or **–w** options are given, **pax** will list the contents of the specified archive. In this mode, **pax** lists normal files one per line, hard link pathnames as

*pathname == linkname*

and symbolic link pathnames (if supported by the implementation) as

*pathname –> linkname*

where *pathname* is the name of the file being extracted, and *linkname* is the name of a file which appeared earlier in the archive.

If the **–v** option is specified, then **pax** list normal pathnames in the same format used by **ls**(1V) with the **–l** option. Hard links are shown as

*<ls –l listing> == linkname*

and symbolic links (if supported) are shown as

*<ls –l listing> –> linkname*

**pax** is capable of reading and writing archives which span multiple physical volumes. Upon detecting an end of medium on an archive which is not yet completed, **pax** will prompt the user for the next volume of the archive and will allow the user to specify the location of the next volume.

When writing to an archive, the standard input is used as a list of pathnames if no *pathname* operands are specified. The format is one pathname per line. Otherwise, the standard input is the archive file, which is formatted according to one of the specifications in **Archive/Interchange File format** in POSIX.1, or some other implementation-defined format.

The user ID and group ID of the process, together with the appropriate privileges, affect the ability of **pax** to restore ownership and permissions attributes of the archived files. See *format-reading utility* in **Archive/Interchange File Format** in POSIX.1.

**OPTIONS**

**–w** Write the files and directories specified by *pathname* to the standard output together with the pathname and status information prescribed by the archive format used. *pathname* refers to the files and (recursively) subdirectories of that directory. If *pathname* is not specified, then the standard input is read to get a list of pathnames to copy, one pathname per line. In this case, only those pathnames appearing on the standard input are copied.

**–r** Read an archive file from the standard input. Only files with names that match *pattern* are selected for extraction. The selected files are conditionally created and copied relative to the current directory tree, subject to the options described below. By default, the owner and group of selected files will be that of the invoking process, and the permissions and modification times will be the sames as those in the archive.

The supported archive formats are automatically detected on input. The default output format is **ustar**, but may be overridden by the **–x** *format* option described below.

**–rw** Read the files and directories named in *pathname* and copy them to the destination *directory*. A directory *pathname* refers to the files and (recursively) subdirectories of that directory. If *pathname* is not specified, the standard input is read to get a list of pathnames to copy, one pathname per line. In this case, only those pathnames appearing on the standard input are copied. The directory named by *directory* must exist and have the proper permissions before the copy can occur.

**–a** Append the files specified by *pathname* to the specified archive.

**–c** Complement the match sense of *pattern*.

**–d** Intermediate directories not explicitly listed in the archive are not created. This option is ignored unless the **–r** option is specified.

**–i** Interactively rename files. Substitutions specified by the **–s** option (described below) are performed before requesting the new file name from the user. A file is skipped if an empty line is entered and **pax** exits with an exit status of 0 if an EOF is encountered.

**–l** Link files rather than copy when possible.

**–m** File modification times are not retained.

**–n** When **–r** is specified, but **–w** is not, the *pattern* arguments are treated as ordinary file names. Only the first occurrence of each of these files in the input archive is read. The **pax** utility exits with a zero exit status after all files in the list have been read. If one or more files in the list is not found, **pax** writes a diagnostic to standard error for each of the files and exits with a non-zero exit status. The file names are compared before any of the **–i**, **–s**, or **–y** options are applied.

**–o** Restore file ownership as specified in the archive. The invoking process must have appropriate privileges to accomplish this.

**–p** Preserve the access time of the input files after they have been copied.

**–u** Copy each file only if it is newer than a pre-existing file with the same name. This implies the **–a** option.

**–v** List file names as they are encountered. Produces a verbose table of contents listing on the standard output when both the **–r** and **–w** options are omitted, otherwise the file names are printed to standard error as they are encountered in the archive.

**–y** Interactively prompt for the disposition of each file. Substitutions specified by **–s** options (described above) are performed before prompting the user for disposition. **pax** exits after receiving an EOF or an input line starting with the character **q**. Files are ignored if an input line starting with anything other than **y** is specified. This option cannot be used in conjunction with the **–i** option.

**–b** *blocking*

Block the output at *blocking* bytes per write to the archive file. A **k** suffix multiplies *blocking* by 1024, a **b** suffix multiplies *blocking* by 512 and a **m** suffix multiplies *blocking* by 1048576 (1 megabyte). If not specified, *blocking* is automatically determined on input and is ignored if the **–rw** option is specified.

**–f** *archive*

Specify the pathname of the input or output archive, overriding the default of the standard input for the **–r** option or the standard output for the **–w** option.

**–s** *replstr*

File names are modified according to the substitution expression using the syntax of **ed**(1) as shown:

–s */old/new/*[**gp**]

Any non-null character may be used as a delimiter (a '/' is used here as an example). Multiple **–s** expressions may be specified; the expressions are applied in the order specified, terminating with the first successful substitution. The optional trailing **g** replaces the *old* expression each time it occurs in the source string. The optional trailing **p** lists successful mappings on the standard error. Files that substitute to an empty string are ignored both on input and output.

**–t** *device*

The *device* option argument is an implementation-defined identifier that names the input or output archive device, overriding the default of the standard input for **–r** and the standard output for **–w**.

**–x** *format*

Specifies the output archive *format*. The input format, which must be one of the following, is automatically determined when the **–r** option is used. The supported formats are:

*cpio* The extended *CPIO* interchange format specified in **Extended CPIO Format in** *POSIX.1*.

*tar* The extended *TAR* interchange format specified in **Extended TAR Format in** *POSIX.1*. This is the default archive format.

Only the last of multiple **–f** or **–t** options take effect.

The options **–a**, **–c**, **–d**, **–i**, **–l**, **–p**, **–t**, **–u**, and **–y** are provided for functional compatibility with the historical *cpio* and *tar* utilities. The option defaults were chosen based on the most common usage of these options, therefore, some of the options have meanings different than those of the historical commands.

**EXAMPLES**

The following command:

**pax –w –f /dev/rmt0 .**

copies the contents of the current directory to tape drive 0.

The commands:

**mkdir** *newdir*
**cd** *olddir*
**pax –rw .** *newdir*

copies the contents of *olddir* to *newdir*.

The command:

**pax –r –s ',//*usr//*,,' –f pax.out**

reads the archive **pax.out** with all files rooted in **/usr** in the archive extracted relative to the current directory.

**FILES**

| | |
|---|---|
| **/dev/tty** | used to prompt the user for information when the **–i** or **–y** options are specified |

**SEE ALSO**

**cpio**(1), **ed**(1), **find**(1), **ls**(1V), **paxcpio**(1V), **tar**(1), **ustar**(1V), **cpio**(5), **tar**(5)

*IEEE Std. 1003.1-1988*

**DIAGNOSTICS**

**pax** will terminate immediately, without processing any additional files on the command line or in the archive.

**EXIT CODES**

**pax** exits with one of the following values:

| | |
|---|---|
| 0 | All files in the archive were processed successfully. |
| >0 | **pax** aborted due to errors encountered during operation. |

**BUGS**

Special permissions may be required to copy or extract special files.

For device, user ID, and group ID numbers larger than 65535 additional header records are output. These records are ignored by some historical version of **cpio**(1) and **tar**(1).

The archive formats described in **Archive/Interchange File Format** have certain restrictions that have been carried over from historical usage. For example, there are restrictions on the length of pathnames stored in the archive.

When getting an '**ls –l**' style listing on **tar** format archives, link counts are listed as zero since the **ustar** archive format does not keep link count information.

**AUTHOR**

Mark H. Colburn
NAPS International
117 Mackubin Street, Suite 1
St. Paul, MN 55102

Sponsored by **The USENIX Association** for public distribution.

## NAME

paxcpio – copy file archives in and out

## SYNOPSIS

**/usr/5bin/paxcpio –o [ aBcv ]**
**/usr/5bin/paxcpio –i [ Bcdfmrtuv ] [** *pattern ...* **]**
**/usr/5bin/paxcpio –p [ adlmruv ]** *directory*

## AVAILABILITY

**paxcpio** is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**paxcpio** produces and reads files in the format specified by the **cpio Archive/Interchange File Format** specified in *IEEE Std. 1003.1-1988*.

When specified with the **–i** argument, **paxcpio** extracts files from the standard input, which is assumed to be the product of a previous '**paxcpio –o**'. Only files with names that match *pattern* are selected. *pattern* is a simple regular expression given in the name-generating notation of the shell. Multiple *pattern*s may be specified, and if no *pattern* is specified, the default is '*****', selecting all files. The extracted files are conditionally created and copied into the current directory, and possibly any levels below, based upon the options specified. The permissions of the files will be those of the previous '**paxcpio –o**'. The owner and group of the files will be that of the current user unless the user has appropriate privileges, in which case **paxcpio** retains the owner and group of the files of the previous '**paxcpio -o**'.

When specified with the **–p** argument, **paxcpio** reads the standard input to obtain a list of files that are conditionally created and copied into the destination *directory* based upon the options described below.

If an error is detected, the cause is reported and **paxcpio** continues to copy other files. **paxcpio** will skip over any unrecognized files that it encounters in the archive.

The following restrictions apply to **paxcpio**:

- Pathnames are restricted to 256 characters.
- Appropriate privileges are required to copy special files.
- Blocks are reported in 512-byte quantities.

## OPTIONS

**–a** Reset access times of input files after they have been copied. When the **–l** option is also specified, the linked files do not have their access times reset. Can only be used with the **–i** or **–o** arguments.

**–B** Input/output is to be blocked 5120 bytes to the record. Can only be used with the **–i** or **–o** arguments for data that is directed to or from character special files.

**–c** Write header information in ASCII character for for portability. Can only be used with the **–i** or **–o** arguments. Note: this option should always be used to write portable files.

**–d** Create directories as needed. Can only be used with the **–i** or **–p** arguments.

**–f** Copy in all files except those in *pattern*s. Can only be used with the **–i** argument.

**–l** Whenever possible, link files rather than copying them. Can only be used with the **–p** argument.

**–m** Retain previous modification times. This option is ineffective on directories that are being copied. Can only be used with the **–i** or **–p** arguments.

**–r** Interactively rename files. The user is asked whether to rename *pattern* each invocation. Read and write permissions for **/dev/tty** are required for this option. If the user types a null line, the file is skipped. Should only be used with the **–i** or **–o** arguments.

**–t** Print a table of contents of the input. No files are created. Can only be used with the **–i** argument.

**–u** Copy files unconditionally; usually an older file will not replace a new file with the same name. Can only be used with the **–i** or **–p** arguments.

**–v** Verbose. Print the names of the affected files. Can only be used with the **–i** argument. Provides a detailed listing when used with the **–t** option.

## EXAMPLES

The following command:

**ls | paxcpio –o > ../newfile**

copies out the files listed by **ls**(1V) and redirects them to the file **newfile.**

The following command:

**cat newfile | paxcpio –id "memo/al" "memo/b*"**

uses the output file **newfile** from **paxcpio –o**, takes those files that match the patterns **"memo/al"** and **"memo/b*"**, creates the directories below the current directory, and places the files in the appropriate directories.

The command:

**find . –depth –print | paxcpio –pdlmv newdir**

takes the file names piped to it from **find**(1) and copies or links those files to another directory named **newdir**, while retaining the modification time.

## EXIT CODES

**paxcpio** exits with one of the following values:

0 All input files were copied.

2 **paxcpio** encountered errors in copying or accessing files or directories. An error will be reported for nonexistent files or directories, or permissions that do not allow the user to access the source or target files.

## FILES

**/dev/tty** used to prompt the user for information when the **–i** or **–r** options are specified

## SEE ALSO

**cpio**(1), **find**(1), **pax**(1V), **tar**(1), **ustar**(1V), **cpio**(5), **tar**(5)

*IEEE Std. 1003.1-1988*

## NOTES

It is important to use the **–depth** option of **find**(1) to generate pathnames for **paxcpio**. This eliminates problems **paxcpio** could have trying to create files under read-only directories.

## AUTHOR

Mark H. Colburn
NAPS International
117 Mackubin Street, Suite 1
St. Paul, MN 55102

Sponsored by **The USENIX Association** for public distribution.

## NAME

perfmeter – display system performance values in a meter or strip chart

## SYNOPSIS

**perfmeter** [ **–s** *sample-time* ] [ **–h** *h-hand-int* ] [ **–m** *m-hand-int* ] [ **–M** *smax minmax maxmax* ] [ **–v** *value* ] [ *hostname* ]

## AVAILABILITY

This command is available with the *SunView User's* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**perfmeter** is a SunView utility that displays performance values (statistics) for a given *hostname*. If no host is specified, statistics on the current host are metered. The **rstatd**(8C) daemon must be running on the machine being metered.

When open, **perfmeter** displays a performance value in the form of a strip chart. When closed, it displays a meter dial. By default, the display is updated with a *sample-time* of 2 seconds. The hour hand of the meter represents the average over a 20-second interval; the minute hand, the average over 2 seconds. The default value displayed is the percent of CPU being utilized.

The maximum scale value for the strip chart will automatically double or halve to accommodate increasing or decreasing values for the host machine. This scale can be restricted to a certain range with the **–M** option.

## OPTIONS

**–s** *sample-time*
: Set the sample time to *sample-time* seconds.

**–h** *h-hand-int*
: Set the hour-hand interval to *h-hand-int* seconds.

**–m** *m-hand-int*
: Set the minute hand interval to *m-hand-int* seconds.

**–M** *smax minmax maxmax*
: Set a range of maximum values for the strip chart. Values for each of the arguments should be powers of 2. *smax* sets the starting maximum-value. *minmax* sets the lowest allowed maximum-value for the scale. *maxmax* sets the highest allowed maximum-value.

**–v** *value*
: Set the performance value to be monitored to one of:

| | |
|---|---|
| **cpu** | Percent of CPU being utilized. |
| **pkts** | Ethernet packets per second. |
| **page** | Paging activity in pages per second. |
| **swap** | Jobs swapped per second. |
| **intr** | Number of device interrupts per second. |
| **disk** | Disk traffic in transfers per second. |
| **cntxt** | Number of context switches per second. |
| **load** | Average number of runnable processes over the last minute. |
| **colls** | Collisions per second detected on the ethernet. |
| **errs** | Errors per second on receiving packets. |

## USAGE

### Commands

You can change the statistic being displayed by clicking the RIGHT mouse button, and then choosing the desired menu item. The other meter parameters can be modified by moving the pointer onto the tool (either open or closed), and typing:

| | |
|---|---|
| **m** | Decrease *minutehandintv* by one |
| **M** | Increase *minutehandintv* by one |
| **h** | Decrease *hourhandintv* by one |
| **H** | Increase *hourhandintv* by one |
| **1–9** | Set *sampletime* to a range from 1 to 9 seconds. |

## FILES

| | |
|---|---|
| **/etc/inetd.conf** | starts statistics server |

## SEE ALSO

**sunview**(1), **netstat**(8C), **rstatd**(8C), **vmstat**(8)

**NAME**

pg – page through a file on a soft-copy terminal

**SYNOPSIS**

**/usr/5bin/pg** [ **–cefns** ] [ *–number* ] [ **–p string** ] [ *+linenumber* ] [ *+/pattern/* ] [ **filename** ... ]

**AVAILABILITY**

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**pg** is a filter that allows you to page through *filename*, one screenful at a time, on a soft-copy terminal. With a *filename* of '–', or no *filename* specified, **pg** reads from the standard input. Each screenful is followed by a prompt. If the user types a RETURN, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, **pg** scans the **terminfo**(5V) data base for the terminal type specified by the environment variable TERM. If TERM is not defined, the terminal type **dumb** is assumed.

The responses that may be typed when **pg** pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1) NEWLINE or SPACE
: Display one page. The address is specified in pages.

(+1) **l** With a relative address **pg** will simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
: Simulate scrolling half a screen forward or backward.

The following perusal commands take no *address*.

**.** or **^L** Redisplay the current page of text.

**$** Display the last full window in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in **ed**(1) are available. They must always be terminated by a NEWLINE, even if the **–n** option is specified.

*i/pattern/*
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*
*i?pattern?*
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the **?**.

After searching, **pg** will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of **pg** can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w** Display another window of text. If *i* is present, set the window size to *i*.

**s** *filename* Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a NEWLINE, even if the **–n** option is specified.

**h** Help by displaying an abbreviated summary of available commands.

**q** or **Q** Quit **pg**.

**!***command* *command* is passed to the shell, whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a NEWLINE, even if the **–n** option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key, normally CTRL-\ or the BREAK (interrupt) key. This causes **pg** to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then **pg** acts just like **cat**(1V), except that a header is printed before each file (if there is more than one).

**OPTIONS**

The command line options are:

*–number* An integer specifying the size (in lines) of the window that **pg** is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

**–p** *string* Use **string** as the prompt. If the prompt string contains a '**%d**', the first occurrence of '**%d**' in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is '**:**'.

**–c** Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the **terminfo**(5V) data base.

**–e** Do *not* pause at the end of each file.

**–f** Inhibit **pg** from splitting lines. Normally, **pg** splits lines longer than the screen width, but some sequences of characters in the text being displayed (for instance, escape sequences for underlining) generate undesirable results.

**–n** Automatic end of command as soon as a command letter is entered. Normally, commands must be terminated by a NEWLINE character.

**–s** Print all messages and prompts in standout mode (usually inverse video).

*+linenumber* Start up at *linenumber*.

*+/pattern/* Start up at the first line containing the regular expression pattern.

**EXAMPLES**

A sample usage of **pg** in reading system news would be

```
news | pg -p '(Page %d):'
```

**FILES**

| | |
|---|---|
| **/usr/share/lib/terminfo/*** | terminal information data base |
| **/tmp/pg*** | temporary file when input is from a pipe |

**SEE ALSO**

**cat**(1V), **crypt**(1), **ed**(1), **grep**(1V), **more**(1), **terminfo**(5V)

**BUGS**

If terminal TAB characters are not set every eight positions, undesirable results may occur.

When using **pg** as a filter with another command that changes the terminal I/O options (for instance, **crypt**(1)), terminal settings may not be restored correctly.

**NOTES**

While waiting for terminal input, **pg** responds to BREAK , DEL , and ^ by terminating execution. Between prompts, however, these signals interrupt **pg**'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of **more**(1) will find that the **z** and **f** commands are available, and that the terminal '/', '^', or '?' may be omitted from the searching commands.

## NAME

plot, aedplot, bgplot, crtplot, dumbplot, gigiplot, hpplot, implot, t300, t300s, t4013, t450, tek – graphics filters for various plotters

## SYNOPSIS

**plot** [ **–T***terminal* ]

## AVAILABILITY

This command is available with the *Graphics* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**plot** reads plotting instructions (see **plot**(5)) from the standard input and produces plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* is specified, the environment variable **TERM** is used. The default *terminal* is **tek**.

## ENVIRONMENT

Except for **ver**, the following terminal-types can be used with '**lpr –g**' see **lpr**(1)) to produce plotted output:

**2648 | 2648a | h8 | hp2648 | hp2648a**
Hewlett Packard 2648 graphics terminal.

**300** DASI 300 or GSI terminal (Diablo mechanism).

**300s | 300S** DASI 300s terminal (Diablo mechanism).

**450** DASI Hyterm 450 terminal (Diablo mechanism).

**4013** Tektronix 4013 storage scope.

**4014 | tek** Tektronix 4014 and 4015 storage scope with Enhanced Graphics Module. (Use 4013 for Tektronix 4014 or 4015 without the Enhanced Graphics Module).

**aed** AED 512 color graphics terminal.

**bgplot | bitgraph**
BBN bitgraph graphics terminal.

**crt** Any crt terminal capable of running **vi**(1).

**dumb | un | unknown**
Dumb terminals without cursor addressing or line printers.

**gigi | vt125** DEC vt125 terminal.

**h7 | hp7 | hp7221**
Hewlett Packard 7221 graphics terminal.

**implot** Imagen plotter.

**var** Benson Varian printer-plotter

**ver** Versatec D1200A printer-plotter. The output is scan-converted and suitable input to '**lpr –v**'.

## FILES

**/usr/bin/t300**
**/usr/bin/t300s**
**/usr/bin/t4013**
**/usr/bin/t450**
**/usr/bin/aedplot**
**/usr/bin/crtplot**
**/usr/bin/gigiplot**
**/usr/bin/implot**

**/usr/bin/plot**
**/usr/bin/bgplot**
**/usr/bin/dumbplot**
**/usr/bin/hpplot**
**/usr/bin/vplot**
**/usr/bin/tek**
**/usr/bin/t450**
**/usr/bin/t300**
**/usr/bin/t300s**
**/usr/bin/vplot**
**/var/tmp/vplot***nnnnnn*

**SEE ALSO**

**graph**(1G), **lpr**(1), **vi**(1), **plot**(3X), **plot**(5), **rasterfile**(5)

## NAME

pr – prepare file(s) for printing, perhaps in multiple columns

## SYNOPSIS

**pr** [ −l+ *n* ] [ −**fmt** ] [ −**h** *string* ] [ −**l***n* ] [ −**s***c* ] [ −**w***n* ] [ *filename* ] ...

## SYSTEM V SYNOPSIS

**/usr/5bin/pr** [ −l+ *n* ] [ −**adfmprt** ] [ −**e***ck* ] [ −**h** *string* ] [ −**i***ck* ] [ −**l***n* ] [ −**n***ck* ] [ −**o***n* ] [ −**s***c* ] [ −**w***n* ] [ *filename* ] ...

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**pr** prepares one or more *filename*s for printing. By default, the output is separated into pages headed by a date, the name of the file, and the page number. **pr** prints its standard input if there are no *filename* arguments. FORMFEED characters in the input files cause page breaks in the output, as expected.

By default, columns are of equal width, separated by at least one SPACE; lines that do not fit are truncated. If the −**s** option is used, lines are not truncated and columns are separated by the separation character.

Inter-terminal messages using **write**(1) are forbidden during a **pr**.

## OPTIONS

Options apply to all following *filename*s but may be reset between *filename*s:

**−f** Use FORMFEED characters instead of NEWLINE characters to separate pages. A FORMFEED is assumed to use up two blank lines at the top of a page. Thus this option does not affect the effective page length.

**−m** Print all *filename*s simultaneously, each in one column, for example:

| | | |
|---|---|---|
| **Print** | **Print** | **The** |
| **the** | **the** | **third** |
| **lines** | **lines** | **file's** |
| **of** | **of** | **lines** |
| **file** | **file** | **go** |
| **one.** | **two.** | **here.** |

**−t** Do not print the 5-line header or the 5-line trailer normally supplied for each page. Pages are not separated when this option is used, even if the −**f** option was used. The −**t** option is intended for applications where the results should be directed to a file for further processing.

**−h** *string*
Use *string*, instead of the file name, in the page header.

**−l***n* Take the length of the page to be *n* lines instead of the default 66.

**−s***c* Separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a TAB.

**−w***n* For multicolumn output, take the width of the page to be *n* characters instead of the default 72.

**−***n* Produce *n*-column output. For example:

| | | |
|---|---|---|
| **Print** | **of** | **in** |
| **the** | **one** | **three** |
| **lines** | **file** | **columns.** |

Columns are not balanced; if, for example, there are as many lines in the file as there are lines on the page, only one column will be printed. Even if the −**t** option (see below) is specified, blank lines will be printed at the end of the output to pad it to a full page.

**+***n* Begin printing with page *n*.

SYSTEM V OPTIONS

When the *–n* option is specified for multicolumn output, columns are balanced. For example, if there are as many lines in the file as there are lines to be printed, and two columns are to be printed, each column will contain half the lines of the file. If the **–t** option is specified, no blank lines will be printed to pad the last page.

The options **–e** and **–i** are assumed for multicolumn output. The **–m** option overrides the **–k** and **–a** options.

The **–f** option does not assume that FORMFEED uses up two blank lines; blank lines will be printed after the FORMFEED if necessary. If the standard output is a terminal, **–f** will cause **pr** to wait for a RETURN before printing the first page.

**–a** When combined with the *–n* option, print multicolumn output across the page. For example:

| | | |
|---|---|---|
| **Print** | **the** | **lines** |
| **of** | **one** | **file** |
| **in** | **three** | **columns.** |

**–d** Double-space the output.

**–p** Pause before beginning each page if the output is directed to a terminal (**pr** will ring the bell at the terminal and wait for a RETURN).

**–r** Do Not print diagnostic reports if a file cannot be opened, or if it is empty.

**–e***ck* Expand *input* TAB characters to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default TAB settings at every eighth position are assumed. TAB characters in the input are expanded into the appropriate number of SPACE characters. If *c* (any non-digit character) is given, it is treated as the input TAB character (default for *c* is the TAB character).

**–i***ck* In *output*, replace white space wherever possible by inserting TAB characters to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default TAB settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output TAB (default for *c* is the TAB character).

**–n***ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **–m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a TAB).

**–o***k* Offset each line by *k* character positions. The number of character positions per line is the sum of the width and offset.

EXAMPLES

Print a file called **dreadnought** on the printer — this is the simplest use of **pr**:

```
example% pr dreadnought | lpr
example%
```

Produce three laminations of a file called **ridings** side by side in the output, with no headers or trailers, the results to appear in the file called **Yorkshire**:

```
example% pr –m –t ridings ridings ridings > Yorkshire
example%
```

FILES

**/dev/tty*** to suspend messages.

SEE ALSO

**cat**(1V), **lpr**(1), **write**(1)

DIAGNOSTICS

**can't print 0 cols, using 1**
–**0** was specified as a –*n* option.

**pr: bad key** *key*
An illegal option was given.

**pr: No room for columns.**
The number of columns requested will not fit on the page.

**pr: Too many args**
More than 10 files were specified with the –**m** option.

*filename* **:** *error*
*filename* could not be opened. This diagnostic is not printed if **pr** is printing on a terminal.

SYSTEM V DIAGNOSTICS

**pr: bad option**
An illegal option was given.

**pr: width too small**
The number of columns requested will not fit on the page.

**pr: too many files**
More than 10 files were specified with the –**m** option.

**pr: page-buffer overflow**
The formatting required is more complicated than **pr** can handle.

**pr: out of space**
**pr** could not allocate a buffer it required.

**pr:** *filename***-- empty file**
*filename* was empty. This diagnostic is printed after all the files are printed if **pr** is printing on a terminal.

**pr: can't open** *filename*
*filename* could not be opened. This diagnostic is printed after all the files are printed if **pr** is printing on a terminal.

BUGS

The options described above interact with each other in strange and as yet to be defined ways.

**NAME**

printenv – display environment variables currently set

**SYNOPSIS**

**printenv** [ *variable* ]

**DESCRIPTION**

**printenv** prints out the values of the variables in the environment. If a *variable* is specified, only its value is printed.

**SEE ALSO**

**csh**(1), **sh**(1), **stty**(1V), **tset**(1), **environ**(5V)

**DIAGNOSTICS**

If a *variable* is specified and it is not defined in the environment, **printenv** returns an exit status of **1**.

## NAME

prof – display profile data

## SYNOPSIS

**prof** [ **–alnsz** ] [ **–v** *–low* [ *–high* ] ] [ *image-file* [ *profile-file* ... ] ]

## DESCRIPTION

**prof** produces an execution profile of a program. The profile data is taken from the profile file which is created by programs compiled with the **–p** option of **cc**(1V) and other compilers. That option also links in versions of the library routines (see **monitor**(3)) which are compiled for profiling. The symbol table in the executable image file *image-file* (**a.out** by default) is read and correlated with the profile file *profile-file* (**mon.out** by default). For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call. If more than one profile file is specified, the **prof** output shows the sum of the profile information in the given profile files.

To tally the number of calls to a routine, the modules that make up the program must be compiled with the '**cc –p**' option (see **cc**(1V)). This option also means that the profile file is produced automatically.

A single function may be split into subfunctions for profiling by means of the **MARK** macro (see **prof**(3)).

Beware of quantization errors.

The profiled program must call **exit**(2V) or return normally for the profiling information to be saved in the **mon.out** file.

## OPTIONS

**–a** Report all symbols rather than just external symbols.

**–l** Sort the output by symbol value.

**–n** Sort the output by number of calls.

**–s** Produce a summary profile file in **mon.sum**. This is really only useful when more than one profile file is specified.

**–z** Display routines which have zero usage (as indicated by call counts and accumulated time).

**–v** [ *–low* [ *–high* ]]
Suppress all printing and produce a graphic version of the profile on the standard output for display by the **plot**(1G) filters. When plotting, the numbers *low* and *high*, (by default 0 and 100), select a percentage of the profile to be plotted, with accordingly higher resolution.

## ENVIRONMENT

**PROFDIR**
If this environment variable contains a value, place profiling output within that directory, in a file named *pid.programname*. *pid* is the process ID, and *programname* is the name of the program being profiled, as determined by removing any path prefix from the **argv[0]** with which the program was called. If the variable contains a NULL value, no profiling output is produced. Otherwise, profiling output is placed in the file **mon.out**.

## FILES

| | |
|---|---|
| **a.out** | executable file containing namelist |
| **$PROFDIR**/*pid.programname* | |
| **mon.out** | profiling output |
| **mon.sum** | summary profile |

## SEE ALSO

**cc**(1V), **gprof**(1), **plot**(1G), **tcov**(1), **exit**(2V), **profil**(2), **monitor**(3)

**BUGS**

**prof** is confused by the FORTRAN compiler which puts the entry points at the bottom of subroutines and functions.

## NAME

ps – display the status of current processes

## SYNOPSIS

**ps** [ [–]**acCegjklnrSuUvwx** ] [ –**tx** ] | [ *num* ] [ *kernel-name* ] [ *c-dump-file* ] [ *swap-file* ]

## DESCRIPTION

**ps** displays information about processes. Normally, only those processes that are running with your effective user ID and are attached to a controlling terminal (see **termio**(4)) are shown. Additional categories of processes can be added to the display using various options. In particular, the –**a** option allows you to include processes that are not owned by you (that do not have your user ID), and the –**x** option allows you to include processes without control terminals. When you specify both –**a** and –**x**, you get processes owned by anyone, with or without a control terminal. The –**r** option restricts the list of processes printed to "running" processes: runnable processes, those in page wait, or those in short-term non-interruptible waits.

**ps** displays the process ID, under PID; the control terminal (if any), under TT; the cpu time used by the process so far, including both user and system time), under TIME; the state of the process, under STAT; and finally, an indication of the COMMAND that is running.

The state is given by a sequence of four letters, for example, '**RWNA**'.

*First letter* indicates the runnability of the process:

- **R** Runnable processes.
- **T** Stopped processes.
- **P** Processes in page wait.
- **D** Processes in non-interruptible waits; typically short-term waits for disk or NFS I/O.
- **S** Processes sleeping for less than about 20 seconds.
- **I** Processes that are idle (sleeping longer than about 20 seconds).
- **Z** Processes that have terminated and that are waiting for their parent process to die a **wait**(2V) ("zombie" processes).

*Second letter* indicates whether a process is swapped out;

- *blank* Represented as a SPACE character, in this position indicates that the process is loaded (in memory).
- **W** Process is swapped out.
- **>** Process has specified a soft limit on memory requirements and has exceeded that limit; such a process is (necessarily) not swapped.

*Third letter* indicates whether a process is running with altered CPU scheduling priority (**nice**(1)):

- *blank* Represented as a SPACE character, in this position indicates that the process is running without special treatment.
- **N** The process priority is reduced,
- **<** The process priority has been raised artificially.

*Fourth letter* indicates any special treatment of the process for virtual memory replacement. The letters correspond to options to the **vadvise**(2) system call. Currently the possibilities are:

- *blank* Represented as a SPACE character, in this position stands for VA_NORM.
- **A** Stands for VA_ANOM. An **A** typically represents a program which is doing garbage collection.
- **S** Stands for VA_SEQL. An **S** is typical of large image processing programs that are using virtual memory to sequentially address voluminous data.

*kernel-name* specifies the location of the system namelist. If the –**k** option is given, *c-dump-file* tells **ps** where to look for the core dump. Otherwise, the core dump is located in the file **/vmcore** and this argument is ignored. *swap-file* gives the location of a swap file other than the default, **/dev/drum**.

**OPTIONS**

Options must all be combined to form the first argument.

**–a** Include information about processes owned by others.

**–c** Display the command name, as stored internally in the system for accounting purposes, rather than the command arguments, which are kept in the process address space. This is more reliable, if less informative, as the process is free to destroy the latter information.

**–C** Display raw CPU time instead of the decaying average in the **%CPU** field.

**–e** Display the environment as well as the arguments to the command.

**–g** Display all processes. Without this option, **ps** prints only "interesting" processes. Processes are deemed to be uninteresting if they are process group leaders. This normally eliminates top-level command interpreters and processes waiting for users to login on free terminals.

**–j** Display a listing useful for job control information, with fields PPID, PID, PGID, SID, TT, TPGID, STAT, UID, TIME, and COMMAND as described below.

With this option, the STAT field has three additional letters:

**C** indicates the process does not want SIGCHLD when a child changes state done to job control.

**E** The process has completed an exec, and the parent can no longer change the process group of this process.

**O** The process is an orphan, with no parent process to handle job control signals.

**–k** Normally, *kernel-name* defaults to **/vmunix**, *c-dump-file* is ignored, and *swap-file* defaults to **/dev/drum**. With the **–k** option in effect, these arguments default to **/vmunix**, **/vmcore**, and **/dev/drum**, respectively.

**–l** Display a long listing, with fields **F, PPID, CP, PRI, NI, SZ, RSS,** and **WCHAN**, as described below.

**–n** Produce numeric output for some fields. In a long listing, the **WCHAN** field is printed numerically rather than symbolically, or, in a user listing, the **USER** field is replaced by a **UID** field.

**–r** Restrict output to "running" processes.

**–S** Display accumulated CPU time used by this process and all of its reaped children.

**–u** Display user-oriented output. This includes fields **USER, %CPU, %MEM, SZ, RSS** and **START** as described below.

**–U** Update a private database where **ps** keeps system information. Include '**ps –U**' in the **/etc/rc** file.

**–v** Display a version of the output describing virtual memory information. This includes fields **RE, SL, PAGEIN, SIZE, RSS, LIM, %CPU** and **%MEM**, described below.

**–w** Use a wide output format (132 columns rather than 80); if repeated, that is, **–ww**, use arbitrarily wide output. This information is used to decide how much of long commands to print.

**–x** Include processes with no controlling terminal.

The following two options are mutually exclusive. When specified, these options must appear immediately following the last option.

**–t***x* Restrict output to processes whose controlling terminal is *x* (which should be specified as printed by **ps**; for example, **t3** for **/dev/tty3**, **tco** for **/dev/console**, **td0** for **/dev/ttyd0**, **t?** for processes with no terminal, etc). This option must be the last one given.

*num* A process number may be given, in which case the output is restricted to that process. This option must also be last, and must appear with no white space between it and the previous option.

**DISPLAY FORMATS**

Fields that are not common to all output formats:

**USER** Name of the owner of the process.

**%CPU** CPU use of the process; this is a decaying average over up to a minute of previous (real) time. Because the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all **%CPU** fields to exceed 100%.

**NI** Process scheduling increment (see **getpriority**(2) and **nice**(3V)).

**SIZE**

**SZ** The combined size of the data and stack segments (in kilobytes)

**RSS** Real memory (resident set) size of the process (in kilobytes).

**LIM** Soft limit on memory used, specified using a call to **getrlimit**(2); if no limit has been specified, this is shown as *xx*.

**%MEM** Percentage of real memory used by this process.

**RE** Residency time of the process (seconds in core).

**SL** Sleep time of the process (seconds blocked).

**PAGEIN** Number of disk I/Os resulting from references by the process to pages not loaded in core.

**UID** Numeric user-ID of process owner.

**PPID** Numeric ID of parent of process.

**SID** Numeric ID of the session to which the process belongs. SID = PGID = PID indicates a session leader.

**PGID** Numeric ID of the process group of the process.

**TPGID** Numeric ID of the process group associated with the terminal specified under TT (distinguished process group, see **termio**(4)).

**CP** Short-term CPU utilization factor (used in scheduling).

**PRI** Process priority (non-positive when in non-interruptible wait).

**START** Time the process was created if today, or the date it was created if before today.

**WCHAN** Event on which process is waiting (an address in the system). A symbol is chosen that classifies the address, unless numeric output is requested (see the **n** flag). In this case, the address is printed in hexadecimal.

**F** Flags (in hex) associated with process as in **<sys/proc.h>**:

| | | |
|---|---|---|
| **SLOAD** | 00000001 | in core |
| **SSYS** | 00000002 | swapper or pager process |
| **SLOCK** | 00000004 | process being swapped out |
| **SSWAP** | 00000008 | save area flag |
| **STRC** | 00000010 | process is being traced |
| **SWTED** | 00000020 | parent has been told that this process stopped |
| **SULOCK** | 00000040 | user can set lock in core |
| **SPAGE** | 00000080 | process in page wait state |
| **SKEEP** | 00000100 | another flag to prevent swap out |
| **SOMASK** | 00000200 | restore old mask after taking signal |
| **SWEXIT** | 00000400 | working on exiting |
| **SPHYSIO** | 00000800 | doing physical I/O |
| **SVFORK** | 00001000 | process resulted from vfork() |
| **SVFDONE** | 00002000 | another vfork flag |
| **SNOVM** | 00004000 | no vm, parent in a vfork() |
| **SPAGI** | 00008000 | init data space on demand, from vnode |

| | | |
|---|---|---|
| **SSEQL** | 00010000 | user warned of sequential vm behavior |
| **SUANOM** | 00020000 | user warned of anomalous vm behavior |
| **STIMO** | 00040000 | timing out during sleep |
| **SORPHAN** | 00080000 | process is orphaned |
| **STRACNG** | 00100000 | process is tracing another process |
| **SOWEUPC** | 00200000 | process is being profiled and has a pending count increment |
| **SSEL** | 00400000 | selecting; wakeup/waiting danger |
| **SFAVORD** | 02000000 | favored treatment in swapout and pageout |
| **SLKDONE** | 04000000 | record-locking has been done |
| **STRCSYS** | 08000000 | tracing system calls |
| **SNOCLDSTOP** | 10000000 | SIGCHLD not sent when child stops |
| **SEXECED** | 20000000 | process has completed an exec |
| **SRPC** | 40000000 | sunview window locking |

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**; a process that is blocked trying to exit is marked **<exiting>**; otherwise, **ps** makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area.

## ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **ps**. On entry to **ps**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## FILES

| | |
|---|---|
| **/vmunix** | system namelist |
| **/dev/kmem** | kernel memory |
| **/dev/drum** | swap device |
| **/vmcore** | core file |
| **/dev** | searched to find swap device and terminal names |
| **/etc/psdatabase** | system namelist, device, and wait channel information |

## SEE ALSO

**kill**(1), **w**(1), **getpriority**(2), **getrlimit**(2), **wait**(2V), **vadvise**(2), **nice**(3V), **termio**(4), **locale**(5), **pstat**(8)

## BUGS

Things can change while **ps** is running; the picture it gives is only a close approximation to the current state.

## NAME

ptx – generate a permuted index

## SYNOPSIS

**ptx** [ **–f** ] [ **–t** ] [ **–w** *n* ] [ **–g** *n* ] [ **–o** *only* ] [ **–i** *ignore* ] [ **–b** *break] [ –***r** ] [ *input* [ *output* ] ]

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**ptx** generates a permuted index of the contents of file *input* onto file *output* (defaults are standard input and output). **ptx** has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. **ptx** produces output in the form:

*.xx "tail" "before keyword" "keyword and after" "head"*

where *xx* may be an **nroff**(1) or **troff**(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *tail* and *head*, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

## OPTIONS

| | |
|---|---|
| **–f** | Fold upper and lower case letters for sorting. |
| **–t** | Prepare the output for the phototypesetter; the default line length is 100 characters. |
| **–w** *n* | Use the next argument, *n*, as the width of the output line. The default line length is 72 characters. |
| **–g** *n* | Use the next argument, *n*, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters. |
| **–o** *only* | Use as keywords only the words given in the *only* file. |
| **–i** *ignore* | Do not use as keywords any words given in the *ignore* file. If the **–i** and **–o** options are missing, use **/usr/lib/eign** as the *ignore* file. |
| **–b** *break* | Use the characters in the *break* file to separate words. In any case, TAB, NEWLINE, and SPACE characters are always used as break characters. |
| **–r** | Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line. |

## FILES

**/usr/lib/eign**

## SEE ALSO

**nroff**(1), **sort**(1V), **troff**(1)

## BUGS

Line length counts do not account for overstriking or proportional spacing.

**NAME**

pwd – display the pathname of the current working directory

**SYNOPSIS**

**pwd**

**DESCRIPTION**

**pwd** prints the pathname of the working (current) directory.

If you are using **csh**(1), you can use the **dirs** builtin command to do the same job more quickly; *but* **dirs** can give a different answer in the rare case that the current directory or a containing directory was moved after the shell descended into it. This is because **pwd** searches back up the directory tree to report the true pathname, whereas **dirs** remembers the pathname from the last **cd**(1) command. The example below illustrates the differences.

```
example% cd /usr/wendy/january/reports
example% pwd
/usr/wendy/january/reports
example% dirs
~/january/reports
example% mv ~/january ~/february
example% pwd
/usr/wendy/february/reports
example% dirs
~/january/reports
example%
```

**pwd** and **dirs** also give different answers when you change directory through a symbolic link. For example:

```
example% cd /usr/wendy/january/reports
example% pwd
/usr/wendy/january/reports
example% dirs
~/january/reports
example% ls –l /usr/wendy/january
lrwxrwxrwx 1 wendy 17 Jan 30 1983 /usr/wendy/january –> /usr/wendy/1984/jan/
example% cd /usr/wendy/january
example% pwd
/usr/wendy/1984/jan
example% dirs
/usr/wendy/january
```

The pathnames of files mounted with the Automounter can also change if the file is not used for a certain time interval (the default is five minutes). To prevent this, set the environment variable AUTOMOUNT_FIXNAMES. See **automount**(8) for more information.

**SEE ALSO**

**cd**(1), **csh**(1), **automount**(8)

**NAME**

quota – display a user's disk quota and usage

**SYNOPSIS**

**quota** [ –v ] [ *username* ]

**DESCRIPTION**

**quota** displays users' disk usage and limits. Only the super-user may use the optional *username* argument to view the limits of users other than himself.

**quota** without options displays only warnings about mounted file systems where usage is over quota. Remotely mounted file systems which are mounted with the "noquota" option (see **fstab**(5)) are ignored.

**OPTIONS**

–v Display user's quotas on all mounted file systems where quotas exist.

**FILES**

| | |
|---|---|
| **quotas** | quota file at the file system root |
| **/etc/mtab** | list of currently mounted filesystems |

**SEE ALSO**

**quotactl**(2), **fstab**(5), **edquota**(8), **quotaon**(8), **rquotad**(8C)

## NAME

ranlib – convert archives to random libraries

## SYNOPSIS

**ranlib** [ **–t** ] *archive...*

## DESCRIPTION

**ranlib** converts each *archive* to a form that can be linked more rapidly. **ranlib** does this by adding a table of contents called **__.SYMDEF** to the beginning of the archive. **ranlib** uses **ar**(1V) to reconstruct the archive. Sufficient temporary file space must be available in the file system that contains the current directory.

## OPTIONS

**–t** **ranlib** only "touches" the archives and does not modify them. This is useful after copying an archive or using the **–t** option of **make**(1) to avoid having **ld**(1) complain about an "out of date" symbol table.

## SEE ALSO

**ar**(1V), **ld**(1), **lorder**(1), **make**(1)

## BUGS

Because generation of a library by **ar** and randomization of the library by **ranlib** are separate processes, phase errors are possible. The linker, **ld**, warns when the modification date of a library is more recent than the creation date of its dictionary; but this means that you get the warning even if you only copy the library.

## NAME

rasfilter8to1 – convert an 8-bit deep rasterfile to a 1-bit deep rasterfile

## SYNOPSIS

**rasfilter8to1** [ **–d** ] [ **–rgba** *threshold* ] [ *infile* [ *outfile* ]]

## DESCRIPTION

**rasfilter8to1** reads the 8-bit deep rasterfile *infile* (the standard input default) and converts it to the 1-bit deep rasterfile *outfile* (standard output default) by thresholding or ordered dither. The output format is Sun standard rasterfile format (see **<rasterfile.h>**). This command is useful for viewing 8-bit rasterfiles on devices that can only display monochrome images.

## OPTIONS

**–d** Use ordered dither to convert the input file instead of thresholding.

**–rgba** *threshold*

Set the threshold for the red, green, blue, and average pixel color values. Pixels whose color values are greater than or equal to all of the thresholds are given a value of 0 (white) in the output rasterfile; other pixels are set to 1 (black). The average threshold defaults to 128, the individual thresholds to zero.

## EXAMPLE

The command

**example% screendump –f /dev/cgtwo | rasfilter8to1 | lpr –Pversatec –v**

prints a monochromatic representation of the **/dev/cgtwo** frame buffer on the printer named **versatec** using the **v** output filter (see **/etc/printcap**). The printer must support an appropriate imaging model such as PostScript in order to print the image.

## FILES

**/usr/lib/rasfilters/*** filters for non-standard rasterfile formats

## SEE ALSO

**lpr**(1), **rastrepl**(1), **screendump**(1), **screenload**(1)

*Pixrect Reference Manual*

## NAME

rastrepl – magnify a raster image by a factor of two

## SYNOPSIS

**rastrepl** [ *input-file* [ *output-file* ]]

## DESCRIPTION

**rastrepl** reads the rasterfile *input-file* (the standard input default) and converts it to the rasterfile *output-file* (the standard output default) which is twice as large in width and height. Pixel replication is used to magnify the image. The output file has the same type as the input file.

## EXAMPLES

The following command:

**example% screendump | rastrepl | lpr –Pversatec –v**

sends a rasterfile containing the current frame buffer contents to the Versatec plotter, doubling the size of the image so that it fills a single page.

## FILES

**/usr/lib/rasfilters/*** filters for non-standard rasterfile formats

## SEE ALSO

**lpr**(1), **screendump**(1), **screenload**(1)

*Pixrect Reference Manual*

## NAME

rcp – remote file copy

## SYNOPSIS

**rcp** [ **–p** ] *filename1 filename2*
**rcp** [ **–pr** ] *filename...directory*

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**rcp** copies files between machines. Each *filename* or *directory* argument is either a remote file name of the form:

*hostname*:*path*

or a local file name (containing no ':' characters, or a '/' before any ':'s).

If a *filename* is not a full path name, it is interpreted relative to your home directory on *hostname*. A *path* on a remote host may be quoted (using \, ", or ') so that the metacharacters are interpreted remotely.

**rcp** does not prompt for passwords; your current local user name must exist on *hostname* and allow remote command execution by **rsh**(1C).

**rcp** handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form

*username*@*hostname*:*filename*

to use *username* rather than your current local user name as the user name on the remote host. **rcp** also supports Internet domain addressing of the remote host, so that:

*username*@*host.domain*:*filename*

specifies the username to be used, the hostname, and the domain in which that host resides. Filenames that are not full path names will be interpreted relative to the home directory of the user named *username*, on the remote host.

The destination hostname may also take the form *hostname.username*:*filename* to support destination machines that are running older versions of **rcp**.

## OPTIONS

**–p** Attempt to give each copy the same modification times, access times, and modes as the original file.

**–r** Copy each subtree rooted at *filename*; in this case the destination must be a directory.

## FILES

**.cshrc**
**.login**
**.profile**

## SEE ALSO

**ftp**(1C), **rlogin**(1C), **rsh**(1C)

## BUGS

**rcp** does not detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

**rcp** can become confused by output generated by commands in a **.profile**, **.cshrc**, or **.login** file on the remote host.

**rcp** requires that the source host have permission to execute commands on the remote host when doing third-party copies.

If you forget to quote metacharacters intended for the remote host you get an incomprehensible error message.

## NAME

rdist – remote file distribution program

## SYNOPSIS

**rdist** [ **–b** ] [ **–h** ] [ **–i** ] [ **–n** ] [ **–q** ] [ **–R** ] [ **–v** ] [ **–w** ] [ **–y** ] [ **–d** *macro* = *value* ]
[ **–f** *distfile* ] [ **–m** *host* ] ... [ *package* ... ]

**rdist** [ **–b** ] [ **–h** ] [ **–i** ] [ **–n** ] [ **–q** ] [ **–R** ] [ **–v** ] [ **–w** ] [ **–y** ]
**–c** *pathname* ... [*login*@]*hostname*[:*destpath*]

## AVAILABILITY

This command is available with the *Networking* software installation option. See *Installing SunOS 4.1* for information about installing optional software.

## DESCRIPTION

**rdist** maintains copies of files on multiple hosts. It preserves the owner, group, mode, and modification time of the master copies, and can update programs that are executing. Normally, a copy on a remote host is updated if its size or modification time differs from the original on the local host. **rdist** reads the indicated *distfile* for instructions on updating files and/or directories. If *distfile* is '–', the standard input is used. If no **–f** option is present, **rdist** first looks in its working directory for **distfile**, and then for **Distfile**, for instructions.

**rdist** updates each *package* specified on the command line; if none are given, all packages are updated according to their entries in the *distfile*.

## OPTIONS

**–b** Binary comparison. Perform a binary comparison and update files if they differ, rather than merely comparing dates and sizes.

**–h** Follow symbolic links. Copy the file that the link points to rather than the link itself.

**–i** Ignore unresolved links. **rdist** will normally try to maintain the link structure of files being transffered and warn the user if all the links cannot be found.

**–n** Print the commands without executing them. This option is useful for debugging a distfile.

**–q** Quiet mode. Do not display the files being updated on the standard output.

**–R** Remove extraneous files. If a directory is being updated, remove files on the remote host that do not correspond to those in the master (local) directory. This is useful for maintaining truly identical copies of directories.

**–v** Verify that the files are up to date on all the hosts. Any files that are out of date are displayed, but no files are updated, nor is any mail sent.

**–w** Whole mode. The whole file name is appended to the destination directory name. Normally, only the last component of a name is used when renaming files. This preserves the directory structure of the files being copied, instead of flattening the directory structure. For instance, renaming a list of files such as ( **dir1/f1 dir2/f2** ) to **dir3** would create files **dir3/dir1/f1** and **dir3/dir2/f2** instead of **dir3/f1** and **dir3/f2**. When the **–w** option is used with a filename that begins with ~, everything except the home directory is appended to the destination name.

**–y** Younger mode. Do not update remote copies that are younger than the master copy, but issue a warning message instead.

**–d** *macro*=*value*
Define *macro* to have *value*. This option is used to define or override macro definitions in the distfile. *value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by white space.

**-c** *pathname ...* [*login*@]*hostname*[:*destpath*]

Update each *pathname* on the named host. (Relative filenames are taken as relative to your home directory.) If the '*login*@' prefix is given, the update is performed with the user ID of *login*. If the ':*destpath*' is given, the remote file is installed as that pathname.

**-f** *distfile*

Use the description file *distfile*. A '-' as the *distfile* argument denotes the standard input.

**-m** *host*

Limit which machines are to be updated. Multiple **-m** arguments can be given to limit updates to a subset of the hosts listed in the distfile.

## USAGE

### Packages

A typical package begins with a label composed of the package name followed by a colon:

*package*:

This label allows you to group any number of file-to-host and file-to-timestamp mappings into a single distribution package. If no package label appears in the distfile, the default package includes all mappings in the file.

A file-to-host mapping specifies a list of files or directories to distribute, their destination host(s), and any **rdist** primitives to use in performing the update. A mapping of this sort takes the form:

( *pathname ...* ) -> ( *hostname ...* ) *primitive* ; [*primitive* ;] ...

In this case, each *pathname* is the full pathname of a local file or directory to distribute; each *hostname* is the name of a remote host on which those files are to be copied, and *primitive* is one of the **rdist** primitive listed under *Primitives*, below. If there is only one *pathname* or *hostname*, the surrounding parentheses can be omitted. A *hostname* can also take the form:

*login*@*hostname*

in which case the update is performed as the user named *login*.

A file-to-timestamp mapping is used to monitor which local files are updated with respect to a local "timestamp" file. This mapping takes the form:

(*filename ...* ) :: *timestamp-file primitive* ; [*primitive* ;] ...

In this case, *timestamp-file* is the name of a file, the modification time of which is compared with each named file on the local host. If a file is newer than *time-stamp-file*, **rdist** displays a message to that effect. If there is only one *filename*, the parentheses can be omitted.

### White Space Characters

NEWLINE, TAB, and SPACE characters are all treated as white space; a mapping continues across input lines until the start of the next mapping: either a single *filename* followed by a '->' or the opening parenthesis of a filename list.

### Comments

Comments begin with # and end with a NEWLINE.

### Macros

**rdist** has a limited macro facility. Macros are only expanded in filename or hostname lists, and in the argument lists of certain primitives. Macros cannot be used to stand for primitives or their options, or the '->' or '::' symbols.

A macro definition is a line of the form:

*macro* = *value*

A macro reference is a string of the form:

**${***macro***}**

although (as with **make**(1)) the braces can be omitted if the macro name consists of just one character.

**Metacharacters**

The shell meta-characters: [, ], {, }, * and ? are recognized and expanded (on the local host only) just as they are with **csh**(1). Metacharacters can be escaped by prepending a backslash.

The ~ character is also expanded in the same way as with **csh**, however, it is expanded separately on the local and destination hosts.

**Filenames**

File names that do not begin with / or ~ are taken to be relative to user's home directory on each destination host. Note that they are *not* relative to the current working directory.

**Primitives**

The following primitives can be used to specify actions **rdist** is to take when updating remote copies of each file.

**install** [ **−b** ] [ **−h** ] [ **−i** ] [ **−R** ] [ **−v** ] [ **−w** ] [ **−y** ] [*newname*]

Copy out-of-date files and directories (recursively). If no **install** primitive appears in the package entry, or if no *newname* option is given, the name of the local file is given to the remote host's copy. If absent from the remote host, parent directories in a filename's path are created. To help prevent disasters, a non-empty directory on a target host is not replaced with a regular file or a symbolic link by **rdist**. However, when using the **−R** option, a non-empty directory is removed if the corresponding filename is completely absent on the master host. The options for **install** have the same semantics as their command line counterparts, but are limited in scope to a particular map. The login name used on the destination host is the same as the local host unless the destination name is of the format *login@host*. In that case, the update is performed under the username *login*.

**notify** *address* ...

Send mail to the indicated TCP/IP *address* of the form:

*user@host*

that lists the files updated and any errors that may have occurred. If an address does not contain a '*@host*' suffix, **rdist** uses the name of the destination host to complete the address.

**except** *filename* ...

Omit from updates the files named as arguments.

**except_pat***pattern* ...

Omit from updates the filenames that match each regular-expression *pattern* (see **ed**(1) for more information on regular expressions. Note that \ and $ characters must be escaped in the distfile. Shell variables can also be used within a pattern, however shell filename expansion is not supported.

**special** [*filename*] ... "*command-line*"

Specify a Bourne shell, **sh**(1) command line to execute on the remote host after each named file is updated. If no *filename* argument is present, the *command-line* is performed for every updated file, with the shell variable **FILE** set to the file's name on the local host. The quotation marks allow *command-line* to span input lines in the distfile; multiple shell commands must be separated by semicolons (;).

The default working directory for the shell executing each *command-line* is the user's home directory on the remote host.

## EXAMPLES

The following sample distfile instructs **rdist** to maintain identical copies of a shared library, a shared-library initialized data file, several include files, and a directory, on hosts named **hermes** and **magus**. On **magus**, commands are executed as root. **rdist** notifies **merlin@druid** whenever it discovers that a local file has changed relative to a timestamp file.

```
HOSTS = ( hermes root@magus )

FILES = ( /usr/local/lib/libcant.so.1.1
        /usrlocal/lib/libcant.sa.1.1 /usr/local/include/{*.h}
        /usr/local/bin )

${FILES} -> ${HOSTS}
        install -R ;
${FILES} :: /usr/local/lib/timestamp
        notify merlin@druid ;
```

## FILES

| | |
|---|---|
| **/tmp/rdist*** | temporary file for update lists |

## SEE ALSO

**csh**(1), **ed**(1), **sh**(1), **stat**(2V)

## DIAGNOSTICS

A complaint about mismatch of **rdist** version numbers may really stem from some problem with starting your shell, for example, you are in too many groups.

## BUGS

Source files must reside or be mounted on the local host.

There is no easy way to have a special command executed only once after all files in a directory have been updated.

Variable expansion only works for name lists; there should be a general macro facility.

**rdist** aborts on files that have a negative modification time (before Jan 1, 1970).

There should be a "force" option to allow replacement of non-empty directories by regular files or sym-links. A means of updating file modes and owners of otherwise identical files is also needed.

## WARNINGS

**root** does not have its accustomed access privileges on NFS mounted file systems. Using **rdist** to copy to such a file system may fail, or the copies may be owned by user "nobody."

## NAME

refer – expand and insert references from a bibliographic database

## SYNOPSIS

**refer** [ **–ben** ] [ **–ar** ] [ **–c***string* ] [ **–k***x* ] [ **–l***m,n* ] [ **–p** *filename* ] [ **–s***keys* ] *filename*...

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**refer** is a preprocessor for **nroff**(1), or **troff**(1), that finds and formats references. The input files (standard input by default) are copied to the standard output, except for lines between '.[' and '.]' command lines, Such lines are assumed to contain keywords as for **lookbib**(1), and are replaced by information from a bibliographic data base. The user can avoid the search, override fields from it, or add new fields. The reference data, from whatever source, is assigned to a set of **troff** strings. Macro packages such as **ms**(7) print the finished reference text from these strings. A flag is placed in the text at the point of reference. By default, the references are indicated by numbers.

When **refer** is used with **eqn**(1), **neqn**, or **tbl**(1), **refer** should be used first in the sequence, to minimize the volume of data passed through pipes.

## OPTIONS

**–b** Bare mode — do not put any flags in text (neither numbers or labels).

**–e** Accumulate references instead of leaving the references where encountered, until a sequence of the form:

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to the same source.

**–n** Do not search the default file.

**–a***r* Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted, all author names are reversed.

**–c***string*
Capitalize (with SMALL CAPS) the fields whose key-letters are in *string*.

**–k***x* Instead of numbering references, use labels as specified in a reference data line beginning with the characters *%x*; By default, *x* is **L**.

**–l***m,n* Instead of numbering references, use labels from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either of *m* or *n* is omitted, the entire name or date, respectively, is used.

**–p** *filename*
Take the next argument as a file of references to be searched. The default file is searched last.

**–s***keys* Sort references by fields whose key-letters are in the *keys* string, and permute reference numbers in the text accordingly. Using this option implies the **–e** option. The key-letters in *keys* may be followed by a number indicating how many such fields are used, with a + sign taken as a very large number. The default is **AD**, which sorts on the senior author and date. To sort on all authors and then the date, for instance, use the options '**–sA+T**'.

**FILES**

| | |
|---|---|
| **/usr/dict/papers** | directory of default publication lists and indexes |
| **/usr/lib/refer** | directory of programs |

**SEE ALSO**

**addbib**(1), **eqn**(1), **indxbib**(1), **lookbib**(1), **nroff**(1), **roffbib**(1), **sortbib**(1), **tbl**(1), **troff**(1)

**NAME**

rev – reverse the order of characters in each line

**SYNOPSIS**

**rev** [*filename*] ...

**DESCRIPTION**

**rev** copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

## NAME

rlogin – remote login

## SYNOPSIS

**rlogin** [ **–L** ] [ **–8** ] [ **–e***c* ] [ **–l** *username* ] *hostname*

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**rlogin** establishes a remote login session from your terminal to the remote machine named *hostname*.

Hostnames are listed in the *hosts* database, which may be contained in the **/etc/hosts** file, the Network Information Service (NIS) *hosts* database, the Internet domain name server, or a combination of these. Each host has one official name (the first name in the database entry), and optionally one or more nicknames. Either official hostnames or nicknames may be specified in *hostname*.

Each remote machine may have a file named **/etc/hosts.equiv** containing a list of trusted hostnames with which it shares usernames. Users with the same username on both the local and remote machine may **rlogin** from the machines listed in the remote machine's **/etc/hosts.equiv** file without supplying a password. Individual users may set up a similar private equivalence list with the file **.rhosts** in their home directories. Each line in this file contains two names: a *hostname* and a *username* separated by a SPACE. An entry in a remote user's **.rhosts** file permits the user named *username* who is logged into *hostname* to rlogin to the remote machine as the remote user without supplying a password. If the name of the local host is not found in the **/etc/hosts.equiv** file on the remote machine, and the local username and hostname are not found in the remote user's .rhosts file, then the remote machine will prompt for a password. Hostnames listed in **/etc/hosts.equiv** and **.rhosts** files must be the official hostnames listed in the hosts database; nicknames may not be used in either of these files.

To counter security problems, the **.rhosts** file must be owned by either the remote user or by root.

The remote terminal type is the same as your local terminal type (as given in your environment **TERM** variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. All echoing takes place at the remote site, so that (except for delays) the remote login is transparent. Flow control using **^S** (CTRL-S) and **^Q** (CTRL-Q) and flushing of input and output on interrupts are handled properly.

## ESCAPES

Lines that you type which start with the tilde character are "escape sequences" (the escape character can be changed using the **–e** options):

**~.** Disconnect from the remote host — this is not the same as a logout, because the local host breaks the connection with no warning to the remote end.

**~susp** Suspend the login session (only if you are using the C shell). **susp** is your "suspend" character, usually **^Z**, (CTRL-Z), see **tty**(1).

**~dsusp** Suspend the input half of the login, but output will still be seen (only if you are using the C shell). **dsusp** is your "deferred suspend" character, usually **^Y**, (CTRL-Y), see **tty**(1).

## OPTIONS

**–L** Allow the **rlogin** session to be run in "litout" mode.

**–8** Pass eight-bit data across the net instead of seven-bit data.

**–e***c* Specify a different escape character, *c*, for the line used to disconnect from the remote host.

**–l** *username*
Specify a different *username* for the remote login. If you do not use this option, the remote username used is the same as your local username.

**FILES**

| | |
|---|---|
| **/usr/hosts/*** | for the *hostname* version of the command |
| **/etc/hosts.equiv** | list of trusted hostnames with shared usernames |
| **~/.rhosts** | private list of trusted hostname/username combinations |

**SEE ALSO**

**rsh**(1C), **stty**(1V), **tty**(1), **ypcat**(1), **hosts**(5), **named**(8C)

**BUGS**

This implementation can only use the TCP network service.
More of the environment should be propagated.

**NOTES**

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

## NAME

rm, rmdir – remove (unlink) files or directories

## SYNOPSIS

**rm** [ – ] [ –**fir** ] *filename*...

**rmdir** *directory*...

## DESCRIPTION

**rm** removes (directory entries for) one or more files. If an entry was the last link to the file, the contents of that file are lost. See **ln**(1V) for more information about multiple links to files.

To remove a file, you must have write permission in its directory; but you do not need read or write permission on the file itself. If you do not have write permission on the file and the standard input is a terminal, **rm** displays the file's permissions and waits for you to type in a response. If your response begins with y the file is deleted; otherwise the file is left alone.

**rmdir** removes each named *directory*. **rmdir** only removes empty directories.

## OPTIONS

– Treat the following arguments as filenames '–' so that you can specify filenames starting with a minus.

**–f** Force files to be removed without displaying permissions, asking questions or reporting errors.

**–i** Ask whether to delete each file, and, under **–r**, whether to examine each directory. Sometimes called the *interactive* option.

**–r** Recursively delete the contents of a directory, its subdirectories, and the directory itself.

## SEE ALSO

**ln**(1V), **su**(1V)

## BUGS

'**rm –r**' removes a directory and its files only if your real user ID has write permission on that directory.

## DIAGNOSTICS

**rm:** *filename*: **No such file or directory**

*filename* does not exist. **rm** will also return false (1) if *filename* was not found.

## WARNING

It is forbidden to remove the file '**..**' to avoid the antisocial consequences of inadvertently doing something like '**rm –r .***'.

## NAME

roffbib – format and print a bibliographic database

## SYNOPSIS

**roffbib** [ **–e** ] [ **–h** ] [ **–m** *filename* ] [ **–n***p* ] [ **–o***list* ] [ **–Q** ] [ **–r***aN* ] [ **–s***N* ] [ **–T***term* ] [ **–V** ] [ **–x** ] [ *filename* ] ...

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**roffbib** prints out all records in a bibliographic database, in bibliography format rather than as footnotes or endnotes. Generally it is used in conjunction with **sortbib**(1):

**example% sortbib database | roffbib**

## OPTIONS

**roffbib** accepts all options understood by **nroff**(1) except **–i** and **–q**.

**–e** Produce equally-spaced words in adjusted lines using full terminal resolution.

**–h** Use output tabs during horizontal spacing to speed output and reduce output character count. TAB settings are assumed to be every 8 nominal character widths.

**–m** *filename*
Prepend the macro file **/usr/share/lib/tmac/tmac.name** to the input files. There should be a space between the **–m** and the macro filename. This set of macros will replace the ones defined in **/usr/share/lib/tmac/tmac.bib.**

**–n***p* Number first generated page *p*.

**–o***list* Print only page numbers that appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; a final *N–* means from page *N* to end.

**–Q** Queue output for the phototypesetter. Page offset is set to 1 inch.

**–r***aN* Set register *a* (one-character) to *N*. The command-line argument **–rN1** will number the references starting at 1.

Four command-line registers control formatting style of the bibliography, much like the number registers of **ms**(7). The flag **–rV2** will double space the bibliography, while **–rV1** will double space references but single space annotation paragraphs. The line length can be changed from the default 6.5 inches to 6 inches with the **–rL6i** argument, and the page offset can be set from the default of 0 to one inch by specifying **–rO1i** (capital O, not zero).

**–s***N* Halt prior to every *N* pages for paper loading or changing (default *N* =1). To resume, enter NEWLINE or RETURN.

**–T***term* Specify *term* as the terminal type.

**–V** Send output to the Versatec. Page offset is set to 1 inch. This option not available on Sun386i systems.

**–x** If abstracts or comments are entered following the **%X** field key, **roffbib** will format them into paragraphs for an annotated bibliography. Several **%X** fields may be given if several annotation paragraphs are desired.

## FILES

**/usr/share/lib/tmac/tmac.bib** file of macros used by **nroff/troff**

**SEE ALSO**

**addbib**(1), **indxbib**(1), **lookbib**(1), **nroff**(1) **refer**(1), **sortbib**(1), **troff**(1)

*Formatting Documents*

**BUGS**

Users have to rewrite macros to create customized formats.

## NAME

rpcgen – RPC protocol compiler

## SYNOPSIS

**rpcgen** *infile*
**rpcgen** [ **–D***name*[=*value*] ] [ **–I** [ **–K** *seconds* ] ] [ **–L** ] [ **–T** ] *infile*
**rpcgen** **–c** | **–h** | **–l** | **–m** | **–t** [ **–o** *outfile* ] [ *infile* ]
**rpcgen** **–s** *transport* [ **–o** *outfile* ] [ *infile* ]

## DESCRIPTION

**rpcgen** generates C code to implement an RPC protocol. The input to **rpcgen** is a language similar to C known as the RPC Language (Remote Procedure Call Language). Information about the syntax of RPC Language is available in the *'rpcgen' Programming Guide* in the *Network Programming* manual.

**rpcgen** is normally used as in the first synopsis where it takes an input file and generates four output files. If the *infile* is named **proto.x**, then **rpcgen** generates a header file in **proto.h**, XDR routines in **proto_xdr.c**, server side stubs in **proto_svc.c**, and client side stubs in **proto_clnt.c**. With the –T option, it also generates the RPC dispatch table in **proto_tbl.i**.

The second synposis provides special features which allow for the creation of more sophisticated RPC servers. These features include support for RPC dispatch tables, and user provided **#defines**. The entries in the RPC dispatch table contain:

- pointers to the service routine corresponding to that procedure
- a pointer to the input and output arguments
- the size of these routines

A server can use the dispatch table to check authorization and then to execute the service routine; a client library may use it to deal with the details of storage management and XDR data conversion.

The other two synopses shown above are used when one does not want to generate all the output files, but only a particular one. Their usage is described in the EXAMPLES section below.

The C-preprocessor, **cpp**(1), is run on the input file before it is actually interpreted by **rpcgen**, so all the **cpp** directives are legal within an **rpcgen** input file. For each type of output file, **rpcgen** defines a special **cpp** symbol for use by the **rpcgen** programmer:

| | |
|---|---|
| **RPC_HDR** | defined when compiling into header files |
| **RPC_XDR** | defined when compiling into XDR routines |
| **RPC_SVC** | defined when compiling into server side stubs |
| **RPC_CLNT** | defined when compiling into client side stubs |
| **RPC_TBL** | defined when compiling into RPC dispatch tables |

In addition, **rpcgen** does a little preprocessing of its own. Any line beginning with '**%**' is passed directly into the output file, uninterpreted by **rpcgen**. For every data type referred to in *infile*, **rpcgen** assumes that there exists a routine with the string '**xdr_**' prepended to the data type. If this routine does not exist in the RPC/XDR library, it must be provided. Providing an undefined data type allows customization of XDR routines.

## OPTIONS

**–c** Compile into XDR routines.

**–D***name*[=*value*]
Define a symbol *name*. Equivalent to the **#define** directive in the source. If no *value* is given, *name* is defined as 1. This option may be called more than once.

**–h** Compile into C data-definitions (a header file). The **–T** option can be used in conjunction to produce a header file which supports RPC dispatch tables.

**–I** Compile support for **inetd**(8C) in the server side stubs. Such servers can be self started or can be started by **inetd**. When the server is self-started, it backgrounds itself by default. A special define symbol **RPC_SVC_FG** can be used to run the server process in foreground, or

alternately the user may just compile it without the **–I** option. If there are no pending client requests, the **inetd** servers exit after 120 seconds (default). The default can be changed with the **–K** option. All the error messages for **inetd** servers are always logged in with **syslog**(3).

**–K** *seconds* If the server was started by **inetd**, specify the time in seconds after which the server should exit if there is no further activity. This option is useful for customization. If *seconds* is 0, the server exits after serving that given request. If *seconds* is –1, the server hangs around for ever after being started by **inetd**. This option is valid only with the **–I** option.

**–l** Compile into client side stubs.

**–L** When the servers are started in foreground, use **syslog**( ) to log the server errors instead of printing them on the standard error.

**–m** Compile into server side stubs, but do not generate a "main" routine. This option is useful for doing callback-routines and for people who need to write their own "main" routine to do initialization. For **inetd** support, they should be compiled with the **–I** option. In such cases, it defines 2 global variables: **_rpcpmstart** and **_rpcfdtype**. The value of **_rpcpmstart** should be 1 or 0 depending upon whether it was started by **inetd** or not. The value of **_rpcfdtype** should be **SOCK_STREAM** or **SOCK_DGRAM** depending upon the type of the connection.

**–o** *outfile* Specify the name of the output file. If none is specified, the standard output is used (**–c**, **–h**, **–l**, **–m**, **–s** and **–t** modes only).

**–s** *transport* Compile into server side stubs for the given transport. The supported transports are **udp** and **tcp**. This option may be called more than once so as to compile a server that serves multiple transports. For **inetd** support, they should be compiled with the **–I** option.

**–t** Compile into RPC dispatch table.

**–T** Generate the code to support RPC dispatch tables.

The options **–c**, **–h**, **–l**, **–m**, **–s** and **–t** are used exclusively to generate a particular type of file, while the options **–D**, **–I**, **–L** and **–T** are global and can be used with the other options.

## EXAMPLES

The following example generates all the five files: **prot.h, prot_clnt.c**, **prot_svc.c**, **prot_xdr.c** and **prot_tbl.i**. The server error messages are logged, instead of being sent to the standard error.

```
example% rpcgen –LT prot.x
```

The following example generates **prot.h**, **prot_clnt.c**, **prot_xdr.c** and **prot_svc.c**. **prot_svc.c** supports server invocation by **inetd**. If the server is started by **inetd**, the server exits after 20 seconds of inactivity.

```
example% rpcgen –I –K 20 prot.x
```

The following example sends the header file (with support for dispatch tables) on the standard output.

```
example% rpcgen –hT prot.x
```

The following example sends the server side stubs file for the transport **tcp** on the standard output.

```
example% rpcgen –s tcp prot.x
```

## SEE ALSO

**cpp**(1), **rpc**(3N), **inetd**(8C)

*'rpcgen' Programming Guide* in *Network Programming*

**BUGS**

The RPC Language does not support nesting of structures. As a work-around, structures can be declared at the top-level, and their name used inside other structures in order to achieve the same effect.

Name clashes can occur when using program definitions, since the apparent scoping does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures and types.

## NAME

rsh – remote shell

## SYNOPSIS

**rsh** [ **–l** *username* ] [ **–n** ] *hostname* [ *command* ]
**rsh** *hostname* [ **–l** *username* ] [ **–n** ] [ *command* ]

*hostname* [ **–l** *username* ] [ **–n** ] [ *command* ]

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**rsh** connects to the specified *hostname* and executes the specified *command*. **rsh** copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; **rsh** normally terminates when the remote command does.

If you omit *command*, instead of executing a single command, **rsh** logs you in on the remote host using **rlogin**(1C).

Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. See EXAMPLES.

Hostnames are given in the *hosts* database, which may be contained in the **/etc/hosts** file, the Network Information Service (NIS) *hosts* database, the Internet domain name database, or some combination of the three. Each host has one official name (the first name in the database entry) and optionally one or more nicknames. Official hostnames or nicknames may be given as *hostname*.

If the name of the file from which **rsh** is executed is anything other than "rsh," **rsh** takes this name as its *hostname* argument. This allows you to create a symbolic link to **rsh** in the name of a host which, when executed, will invoke a remote shell on that host. The **/usr/hosts** directory is provided to be populated with symbolic links in the names of commonly used hosts. By including **/usr/hosts** in your shell's search path, you can run **rsh** by typing **hostname** to your shell.

Each remote machine may have a file named **/etc/hosts.equiv** containing a list of trusted hostnames with which it shares usernames. Users with the same username on both the local and remote machine may **rsh** from the machines listed in the remote machine's **/etc/hosts** file. Individual users may set up a similar private equivalence list with the file **.rhosts** in their home directories. Each line in this file contains two names: a *hostname* and a *username* separated by a SPACE. The entry permits the user named *username* who is logged into *hostname* to use **rsh** to access the remote machine as the remote user. If the name of the local host is not found in the **/etc/hosts.equiv** file on the remote machine, and the local username and hostname are not found in the remote user's **.rhosts** file, then the access is denied. The hostnames listed in the **/etc/hosts.equiv** and **.rhosts** files must be the official hostnames listed in the hosts database; nicknames may not be used in either of these files.

**rsh** will not prompt for a password if access is denied on the remote machine unless the *command* argument is omited.

## OPTIONS

**–l** *username*
: Use *username* as the remote username instead of your local username. In the absence of this option, the remote username is the same as your local username.

**–n**
: Redirect the input of **rsh** to **/dev/null**. You sometimes need this option to avoid unfortunate interactions between **rsh** and the shell which invokes it. For example, if you are running **rsh** and start a **rsh** in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. The **–n** option will prevent this.

The type of remote shell (**sh**, **rsh**, or other) is determined by the user's entry in the file **/etc/passwd** on the remote system.

**EXAMPLES**

The following command appends the remote file **lizard.file** from the machine called **lizard** to the file called **example.file** on the machine called **example.**

**example% rsh lizard cat lizard.file >> example.file**

This example appends the file **lizard.file** on the machine called **lizard** to the file **another.lizard.file** which also resides on the machine called **lizard.**

**example% rsh lizard cat lizard.file ">>" another.lizard.file**

**FILES**

**/etc/hosts**
**/usr/hosts/***
**/etc/passwd**

**SEE ALSO**

**rlogin**(1C), **vi**(1), **ypcat**(1), **hosts**(5), **named**(8C), **rshd**(8C)

**BUGS**

You cannot run an interactive command (such as **vi**(1)); use **rlogin** if you wish to do so.

Stop signals stop the local **rsh** process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

The current local environment is not passed to the remote shell.

Sometimes the **–n** option is needed for reasons that are less than obvious. For example, the command below puts your shell into a strange state.

**example% rsh somehost dd if=/dev/nrmt0 bs=20b | tar xvpBf –**

Evidently, what happens is that the **tar** terminates before the **rsh.** The **rsh** then tries to write into the "broken pipe" and, instead of terminating neatly, proceeds to compete with your shell for its standard input. Invoking **rsh** with the **–n** option avoids such incidents.

Note: this bug occurs only when **rsh** is at the beginning of a pipeline and is not reading standard input. Do not use the **–n** if **rsh** actually needs to read standard input. For example, the following command does not produce the bug.

**example% tar cf – . | rsh sundial dd of=/dev/rmt0 obs=20b**

If you were to use the **–n** in a case like this, **rsh** would incorrectly read from **/dev/null** instead of from the pipe.

**NOTES**

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

**NAME**

rup – show host status of local machines (RPC version)

**SYNOPSIS**

**rup** [ **–hlt** ]
**rup** [ *host*... ]

**AVAILABILITY**

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**rup** gives a status similar to **uptime** for remote machines. It broadcasts on the local network, and displays the responses it receives.

Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below.

When *host* arguments are given, rather than broadcasting **rup** will only query the list of specified hosts.

A remote host will only respond if it is running the **rstatd** daemon, which is normally started up from **inetd**(8C).

**OPTIONS**

**–h** Sort the display alphabetically by host name.

**–l** Sort the display by load average.

**–t** Sort the display by up time.

**FILES**

**/etc/servers**

**SEE ALSO**

**ruptime**(1C), **inetd**(8C), **rstatd**(8C)

**BUGS**

Broadcasting does not work through gateways.

## NAME

ruptime – show host status of local machines

## SYNOPSIS

**ruptime** [ **–alrtu** ]

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**ruptime** gives a status line like **uptime** for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 5 minutes are shown as being down.

Normally, the listing is sorted by host name, but this order can be changed by specifying one of the options listed below.

## OPTIONS

**–a** Count even those users who have been idle for an hour or more.

**–l** Sort the display by load average.

**–r** Reverse the sorting order.

**–t** Sort the display by up time.

**–u** Sort the display by number of users.

## FILES

**/var/spool/rwho/whod.***
data files

## SEE ALSO

**rup**(1C), **rwho**(1C)

## NAME

rusers – who's logged in on local machines (RPC version)

## SYNOPSIS

**rusers** [ **–ahilu** ] [ *host...* ]

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

The **rusers** command produces output similar to **users**(1) and **who**(1), but for remote machines. It broadcasts on the local network, and prints the responses it receives. Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below. When *host* arguments are given, rather than broadcasting **rusers** will only query the list of specified hosts.

The default is to print out a listing in the style of **users**(1) with one line per machine. When the **–l** flag is given, a **rwho**(1C) style listing is used. In addition, if a user has not typed to the system for a minute or more, the idle time is reported.

A remote host will only respond if it is running the **rusersd** daemon, which is normally started up from **inetd**(8C).

## OPTIONS

**–a** Give a report for a machine even if no users are logged on.

**–h** Sort alphabetically by host name.

**–i** Sort by idle time.

**–l** Give a longer listing in the style of **who**(1).

**–u** Sort by number of users.

## FILES

**/etc/inetd.conf**

## SEE ALSO

**rwho**(1C), **users**(1), **who**(1), **inetd**(8C), **rusersd**(8C)

## BUGS

Broadcasting does not work through gateways.

## NAME

rwall – write to all users over a network

## SYNOPSIS

**/usr/etc/rwall** *hostname...*
**/usr/etc/rwall –n** *netgroup...*
**/usr/etc/rwall –h hostname –n netgroup**

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**rwall** reads a message from standard input until EOF. It then sends this message, preceded by the line '**Broadcast Message ...**', to all users logged in on the specified host machines. With the **–n** option, it sends to the specified network groups, which are defined in **netgroup**(5).

A machine can only receive such a message if it is running **rwalld**(8C), which is normally started up from **/etc/inetd.conf** by the daemon **inetd**(8C).

## FILES

**/etc/inetd.conf**

## SEE ALSO

**wall**(1), **netgroup**(5), **inetd**(8C), **rwalld**(8C), **shutdown**(8)

## BUGS

The timeout is fairly short in order to be able to send to a large group of machines (some of which may be down) in a reasonable amount of time. Thus the message may not get through to a heavily loaded machine.

**NAME**

rwho – who's logged in on local machines

**SYNOPSIS**

**rwho** [ **–a** ]

**AVAILABILITY**

The **rwho** service daemon, **rwhod**(8C) must be enabled for this command to return useful results. Refer to **finger**(1), **rup**(1C) and **rusers**(1C) for alternatives.

**AVAILABILITY**

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

The **rwho** command produces output similar to **who**(1), but for all machines on your network. If no report has been received from a machine for 5 minutes, **rwho** assumes the machine is down, and does not report users last known to be logged into that machine.

If a user has not typed to the system for a minute or more, **rwho** reports this idle time. If a user has not typed to the system for an hour or more, the user is omitted from the output of **rwho** unless the **–a** flag is given.

**OPTIONS**

**–a** Report all users whether or not they have typed to the system in the past hour.

**FILES**

**/var/spool/rwho/whod.***

information about other machines

**SEE ALSO**

**finger**(1), **rup**(1C), **ruptime**(1C), **rusers**(1C), **who**(1), **rwhod**(8C)

**BUGS**

Does not work through gateways.

This is unwieldy when the number of machines on the local net is large.

## NAME

sccs – front end for the Source Code Control System (SCCS)

## SYNOPSIS

**sccs** [ **–r** ] [ **–d***rootprefix* ] [ **–p***subdir* ] *subcommand* [ *option* ... ] [ *filename* ... ]

## DESCRIPTION

The **sccs** command is a comprehensive, straightforward front end to the various utility programs of the Source Code Control System (SCCS).

**sccs** applies the indicated *subcommand* to the history file associated with each of the indicated files.

The name of an SCCS history file is derived by prepending the 's.' prefix to the filename of a working copy. The **sccs** command normally expects these 's.files' to reside in an SCCS subdirectory. Thus, when you supply **sccs** with a *filename* argument, it normally applies the subcommand to a file named *s.filename* in the SCCS subdirectory. If *filename* is a pathname, **sccs** looks for the history file in the SCCS subdirectory of that file's parent directory. If *filename* is a directory, however, **sccs** applies the subcommand to every s.file file it contains. Thus, the command:

**sccs get program.c**

would apply the **get** subcommand to a history file named:

**SCCS/s.program.c**

while the command:

**sccs get SCCS**

would apply it to every s.file in the SCCS subdirectory.

Options for the **sccs** command itself must appear before the *subcommand* argument. Options for a given subcommand must appear after the *subcommand* argument. These options are specific to each subcommand, and are described along with the subcommands themselves (see **Subcommands**, below).

### Running Setuid

The **sccs** command also includes the capability to run "setuid" to provide additional protection. However this does not apply to subcommands such as **sccs-admin**(1), since this would allow anyone to change the authorizations of the history file. Commands that would do so always run as the real user.

## OPTIONS

**–r** Run **sccs** with the real user ID, rather than set to the effective user ID.

**–d***rootprefix*

Define the root portion of the pathname for SCCS history files. The default root portion is the current directory. Note: *rootprefix* is prepended to the entire *filename* argument, even if *filename* is an absolute pathname. **–d** overrides any directory specified by the **PROJECTDIR** environment variable (see ENVIRONMENT, below).

**–p***subdir*

Define the (sub)directory within which a history file is expected to reside. SCCS is the default. (See EXAMPLES, below).

## USAGE

### Subcommands

Many of the following **sccs** subcommands invoke programs that reside in **/usr/sccs**. Many of these subcommands accept additional arguments that are documented in the reference page for the utility program the subcommand invokes.

**admin** Modify the flags or checksum of an SCCS history file. Refer to **sccs-admin**(1) for more information about the **admin** utility. While **admin** can be used to initialize a history file, you may find that the **create** subcommand is simpler to use for this purpose.

**cdc** **–r***sid* [ **–y**[*comment*] ]

Annotate (change) the delta commentary. Refer to **sccs-cdc**(1). Note: the **fix** subcommand can be used to replace the delta, rather than merely annotating the existing commentary.

**–r***sid* Specify the SCCS delta ID (SID) to which the change notation is to be added. The SID for a given delta is a number, in Dewey decimal format, composed of two or four fields: the *release* and *level* fields, and for branch deltas, the *branch* and *sequence* fields. For instance, the SID for the initial delta is normally **1.1**.

**–y**[*comment*]
Specify the comment with which to annotate the delta commentary. If **–y** is omitted, **sccs** prompts for a comment. A null *comment* results in an empty annotation.

**check** [**–b**] [**–u**[*username*]]

Check for files currently being edited. Like **info** and **tell**, but returns an exit code, rather than producing a listing of files. **check** returns a non-zero exit status if anything is being edited.

**–b** Ignore branches.

**–u**[*username*]
Only check files being edited by you. When *username* is specified, only check files being edited by that user.

**clean** [**–b**]

Remove everything in the current directory that can be retrieved from an SCCS history. Does not remove files that are being edited.

**–b** Do not check branches to see if they are being edited. '**clean –b**' is dangerous when branch versions are kept in the same directory.

**comb** Generate scripts to combine deltas. Refer to **sccs-comb**(1).

**create** Create (initialize) history files. **create** performs the following steps:

- Renames the original source file to **,program.c** in the current directory.
- Create the history file called **s.program.c** in the SCCS subdirectory.
- Performs an '**sccs get**' on **program.c** to retrieve a read-only copy of the initial version.

**deledit** [**–s**] [**–y**[*comment*]]

Equivalent to an '**sccs delta**' and then an '**sccs edit**'. **deledit** checks in a delta, and checks the file back out again, but leaves the current working copy of the file intact.

**–s** Silent. Do not report delta numbers or statistics.

**–y**[*comment*]
Supply a comment for the delta commentary. If **–y** is omitted, **delta** prompts for a comment. A NULL *comment* results in an empty comment field for the delta.

**delget** [**–s**] [**–y**[*comment*]]

Perform an '**sccs delta**' and then an '**sccs get**' to check in a delta and retrieve read-only copies of the resulting new version. See the **deledit** subcommand for a description of **–s** and **–y**. **sccs** performs a **delta** on all the files specified in the argument list, and then a **get** on all the files. If an error occurs during the **delta**, the **get** is not performed.

**delta** [**–s**] [**–y**[*comment*]]

Check in pending changes. Records the line-by-line changes introduced while the file was checked out. The effective user ID must be the same as the ID of the person who has the file checked out. Refer to **sccs-delta**(1). See the **deledit** subcommand for a description of **–s** and **–y**.

**diffs** [**–C**] [**–c***date-time*] [**–r***sid*] *diff-options*

Compare (in **diff**(1) format) the working copy of a file that is checked out for editing, with a version from the SCCS history. Use the most recent checked-in version by default. The **diffs** subcommand accepts the same options as **diff**, with the exception that the **–c** option to **diff** must be specified as **–C**.

**–C** Pass the **–c** option to **diff**.

**–c***date-time*

Use the most recent version checked in before the indicated date and time for comparison. *date-time* takes the form: *yy*[*mm*[*dd*[*hh*[*mm*[*ss*]]]]]. Omitted units default to their maximum possible values; that is **–c7502** is equivalent to **–c750228235959**.

**–r***sid* Use the version corresponding to the indicated delta for comparison.

**edit** Retrieve a version of the file for editing. **'sccs edit'** extracts a version of the file that is writable by you, and creates a **p.**file in the SCCS subdirectory as lock on the history, so that no one else can check that version in or out. ID keywords are retrieved in unexpanded form. **edit** accepts the same options as **get**, below.

**enter** Similar to **create**, but omits the final **'sccs get'**. This may be used if an **'sccs edit'** is to be performed immediately after the history file is initialized.

**fix –r***sid*

Revise a (leaf) delta. Remove the indicated delta from the SCCS history, but leave a working copy of the current version in the directory. This is useful for incorporating trivial updates for which no audit record is needed, or for revising the delta commentary. **fix** must be followed by a **–r** option, to specify the SID of the delta to remove. The indicated delta must be the most recent (leaf) delta in its branch. Use **fix** with caution since it does not leave an audit trail of differences (although the previous commentary is retained within the history file).

**get** [**–ekmps**] [**–c***date-time*] [**–r***sid*]

Retrieve a version from the SCCS history. By default, this is a read-only working copy of the most recent version; ID keywords are in expanded form. Refer to **sccs-get**(1).

**–e** Retrieve a version for editing. Same as **sccs edit**.

**–k** Retrieve a writable copy but do not check out the file. ID keywords are unexpanded.

**–m** Precede each line with the SID of the delta in which it was added.

**–p** Produce the retrieved version on the standard output. Reports that would normally go to the standard output (delta ID's and statistics) are directed to the standard error.

**–s** Silent. Do not report version numbers or statistics.

**–c***date-time*

Retrieve the latest version checked in prior to the date and time indicated by the *date-time* argument. *date-time* takes the form: *yy*[*mm*[*dd*[*hh*[*mm*[*ss*]]]]].

**–r***sid* Retrieve the version corresponding to the indicated SID.

**help** *message-code*|*sccs-command*
**help stuck**

Supply more information about SCCS diagnostics. **help** displays a brief explanation of the error when you supply the code displayed by an SCCS diagnostic message. If you supply the name of an SCCS command, it prints a usage line. **help** also recognizes the keyword **stuck**. Refer to **sccs-help**(1).

**info** [**–b**] [**–u**[*username*]]
Display a list of files being edited, including the version number checked out, the version to be checked in, the name of the user who holds the lock, and the date and time the file was checked out.

**–b** Ignore branches.

**–u**[*username*]
Only list files checked out by you. When *username* is specified, only list files checked out by that user.

**print** Print the entire history of each named file. Equivalent to an '**sccs prs –e**' followed by an '**sccs get –p –m**'.

**prs** [**–el**] [**–c***date-time*] [**–r***sid*]
Peruse (display) the delta table, or other portion of an s.file. Refer to **sccs-prs**(1).

**–e** Display delta table information for all deltas earlier than the one specified with **–r** (or all deltas if none is specified).

**–l** Display information for all deltas later than, and including, that specified by **–c** or **–r**.

**–c***date-time*
Specify the latest delta checked in before the indicated date and time. The *date-time* argument takes the form: *yy*[*mm*[*dd*[*hh*[*mm*[*ss*]]]]].

**–r***sid* Specify a given delta by SID.

**prt** [**–y**] Display the delta table, but omit the MR field (see **sccsfile**(5) for more information on this field). Refer to **sccs-prt**(1).

**–y** Display the most recent delta table entry. The format is a single output line for each filename argument, which is convenient for use in a pipeline with **awk**(1) or **sed**(1V).

**rmdel** **–r***sid*
Remove the indicated delta from the history file. That delta must be the most recent (leaf) delta in its branch. Refer to **sccs-rmdel**(1).

**sccsdiff** **–r***old-sid* **–r***new-sid diff-options*
Compare two versions corresponding to the indicated SIDs (deltas) using **diff**. Refer to **sccs-sccsdiff**(1).

**tell** [**–b**] [**–u**[*username*]]
Display the list of files that are currently checked out, one filename per line.

**–b** Ignore branches.

**–u**[*username*]
Only list files checked out to you. When *username* is specified, only list files check out to that user.

**unedit** "Undo" the last **edit** or '**get –e**', and return the working copy to its previous condition. **unedit** backs out all pending changes made since the file was checked out.

**unget** Same as **unedit**. Refer to **sccs-unget**(1).

**val** Validate the history file. Refer to **sccs-val**(1).

**what** Display any expanded ID keyword strings contained in a binary (object) or text file. Refer to **what**(1) for more information.

**ENVIRONMENT**

If the environment variable **PROJECTDIR** is set to contain an absolute pathname (beginning with a slash), **sccs** searches for SCCS history files in the directory given by that variable. If **PROJECTDIR** does not begin with a slash, it is taken as the name of a user, and **sccs** searches the **src** or **source** subdirectory of that user's home directory for history files.

**EXAMPLES**

**sccs** converts the command:

**sccs -d/usr/src/include get stdio.h**

to:

**/usr/sccs/get /usr/src/include/SCCS/s.stdio.h**

The command:

**sccs -pprivate get include/stdio.h**

becomes:

**/usr/sccs/get include/private/s.stdio.h**

To initialize the history file for a source file named **program.c**: make the SCCS subdirectory, and then use **'sccs create'**:

```
example% mkdir SCCS
example% sccs create program.c
program.c:
1.1
14 lines
```

After verifying the working copy, you can remove the backup file that starts with a comma:

```
example% diff program.c ,program.c
example% rm ,program.c
```

To check out a copy of **program.c** for editing, edit it, and then check it back in:

```
example% sccs edit program.c
1.1
new delta 1.2
14 lines
example% vi program.c
your editing session
example% sccs delget program.c
comments? clarified cryptic diagnostic
1.2
3 inserted
2 deleted
12 unchanged
1.2
15 lines
```

To retrieve a file from another directory into the current directory:

**example% sccs get /usr/src/sccs/cc.c**

or:

**example% sccs -p/usr/src/sccs/ get cc.c**

To check out all files under SCCS in the current directory:

**example% sccs edit SCCS**

To check in all files currently checked out to you:

**example% sccs delta `sccs tell -u`**

**FILES**

| | |
|---|---|
| **SCCS** | SCCS subdirectory |
| **SCCS/d.***file* | temporary file of differences |
| **SCCS/p.***file* | lock (permissions) file for checked-out versions |
| **SCCS/q.***file* | temporary file |
| **SCCS/s.***file* | SCCS history file |
| **SCCS/x.***file* | temporary copy of the **s.**file |
| **SCCS/z.***file* | temporary lock file |
| **/usr/sccs/*** | SCCS utility programs |

**SEE ALSO**

**awk**(1), **diff**(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **sccs-val**(1), **sed**(1V), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

**BUGS**

There is no **sact** subcommand to invoke **/usr/sccs/sact** (see **sccs-sact**(1)). However, the **info** subcommand performs an equivalent function.

## NAME

sccs-admin, admin – create and administer SCCS history files

## SYNOPSIS

**/usr/sccs/admin** [ **–bhnz** ] [ **–a***username|groupid* ] ... [ **–d***flag* ] ... [ **–e***username|groupid* ] ...
[ **–f***flag* [ *value* ] ] ... [ **–i** [ *filename* ] ] [ **–l a**|*release*[,*release*...] ] [ **–m** *mr-list* ] [ **–r***release* ]
[ **–t** [ *description-file* ] ] [ **–y**[*comment*] ] *s.filename* ...

## DESCRIPTION

**admin** creates or modifies the flags and other parameters of SCCS history files. Filenames of SCCS history files begin with the **'s.'** prefix, and are referred to as **s.**files, or "history" files.

The named **s.**file is created if it does not exist already. Its parameters are initialized or modified according to the options you specify. Parameters not specified are given default values when the file is initialized, otherwise they remain unchanged.

If a directory name is used in place of the *s.filename* argument, the **admin** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error. The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

## OPTIONS

**–b** Force encoding of binary data. Files that contain ASCII NUL or other control characters, or that do not end with a NEWLINE, are recognized as binary data files. The contents of such files are stored in the history file in encoded form. See **uuencode**(1C) for details about the encoding. This option is normally used in conjunction with **–i** to force **admin** to encode initial versions not recognized as containing binary data.

**–h** Check the structure of an existing **s.**file (see **sccsfile**(5)), and compare a newly computed check-sum with one stored in the first line of that file. **–h** inhibits writing on the file; and so nullifies the effect of any other options.

**–n** Create a new SCCS history file.

**–z** Recompute the file check-sum and store it in the first line of the **s.**file. Caution: it is important to verify the contents of the history file (see **sccs-val**(1), and the **print** subcommand in **sccs**(1)), since using **–z** on a truly corrupted file may prevent detection of the error.

**–a***username|groupid*
Add a user name, or a numerical group ID, to the list of users who may check deltas in or out. If the list is empty, any user is allowed to do so.

**–d***flag* Delete the indicated *flag* from the SCCS file. The **–d** option may be specified only for existing **s.**files. See **–f** for the list of recognized flags.

**–e***username|groupid*
Erase a user name or group ID from the list of users allowed to make deltas.

**–f***flag* [*value*]
Set the indicated *flag* to the (optional) *value* specified. The following flags are recognized:

- **b** Enable branch deltas. When **b** is set, branches can be created using the **–b** option of the SCCS **get** command (see **sccs-get**(1)).
- **c***ceil* Set a ceiling on the releases that can be checked out. *ceil* is a number less than or equal to 9999. If **c** is not set, the ceiling is 9999.
- **f***floor* Set a floor on the releases that can be checked out. The floor is a number greater than 0 but less than 9999. If **f** is not set, the floor is 1.
- **d***sid* The default delta number, or SID, to be used by an SCCS **get** command.

**i** Treat the '**No id keywords (ge6)**' message issued by an SCCS **get** or **delta** command as an error rather than a warning.

**j** Allow concurrent updates.

**la**
**l***release*[,*release* ...]
Lock the indicated list of releases against deltas. If **a** is used, lock out deltas to all releases. An SCCS '**get –e**' command fails when applied against a locked release.

**n** Create empty releases when releases are skipped. These null (empty) deltas serve as anchor points for branch deltas.

**q***value* Supply a *value* to which the **%Q%** keyword is to expand when a read-only version is retrieved with the SCCS **get** command.

**m***module*
Supply a value for the module name to which the **%M%** keyword is to expand. If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

**t***type* Supply a value for the module type to which the **%Y%** keyword is to expand.

**v** [*program*]
Specify a validation *program* for the MR numbers associated with a new delta. The optional *program* specifies the name of an MR number validity checking *program*. If this flag is set when creating an SCCS file, the **–m** option must also be used, in which case the list of MRs may be empty.

**–i** [*filename*]
Initialize the history file with text from the indicated file. This text constitutes the initial delta, or set of checked-in changes. If *filename* is omitted, the initial text is obtained from the standard input. Omitting the **–i** option altogether creates an empty **s.**file. You can only initialize one **s.**file with text using **–i**. This option implies the **–n** option.

**–la**
**–l***release*[,*release*...]
Unlock the specified releases so that deltas can be checked in. If **a** is specified, allow deltas to be checked in for all releases.

**–m** [*mr-list*]
Insert the indicated Modification Request (MR) numbers into the commentary for the initial version. When specifying more than one MR number on the command line, *mr-list* takes the form of a quoted, space-separated list. A warning results if the **v** flag is not set or the MR validation fails.

**–r***release*
Specify the release for the initial delta. **–r** may be used only in conjunction with **–i**. The initial delta is inserted into release 1 if this option is omitted. The level of the initial delta is always 1; initial deltas are named 1.1 by default.

**–t** [*description-file*]
Insert descriptive text from the file *description-file*. When **–t** is used in conjunction with **–n**, or **–i** to initialize a new s.file, the *description-file* must be supplied. When modifying the description for an existing file: a **–t** option without a *description-file* removes the descriptive text, if any; a **–t** option with a *description-file* replaces the existing text.

**–y** [*comment*]
Insert the indicated *comment* in the "Comments:" field for the initial delta. Valid only in conjunction with **–i** or **–n**. If **–y** option is omitted, a default comment line is inserted that notes the date and time the history file was created.

**FILES**

| | |
|---|---|
| **s.*** | history file |
| SCCS/**s.*** | history file in SCCS subdirectory |
| **z.*** | temporary lock file |

**WARNINGS**

The last component of all SCCS filenames must have the '**s.**' prefix. New SCCS files are given mode 444 (see **chmod**(1V)). All writing done by **admin** is to a temporary file with an **x.** prefix, created with mode 444 for a new SCCS file, or with the same mode as an existing SCCS file. After successful execution of **admin**, the existing **s.** file is removed and replaced with the **x.**file. This ensures that changes are made to the SCCS file only when no errors have occurred.

It is recommended that directories containing SCCS files have permission mode 755, and that the **s.**files themselves have mode 444. The mode for directories allows only the owner to modify the SCCS files contained in the directories, while the mode of the **s.**files prevents all modifications except those performed using SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner to allow use of a text editor. However, extreme care must be taken when doing this. The edited file should *always* be processed by an '**admin** **–h**' to check for corruption, followed by an '**admin** **–z**' to generate a proper check-sum. Another '**admin** **–h**' is recommended to ensure that the resulting **s.**file is valid.

**admin** also uses a temporary lock **s.**file, starting with the '**z.**' prefix, to prevent simultaneous updates to the **s.**file. See **sccs-get**(1) for further information about the '**z.**file'.

**SEE ALSO**

**sccs**(1), **sccs-cdc**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-rmdel**(1), **sccs-val**(1), **sccsfile**(5)

*Programming Utilities and Libraries.*

**DIAGNOSTICS**

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

## NAME

sccs-cdc, cdc – change the delta commentary of an SCCS delta

## SYNOPSIS

**/usr/sccs/cdc** **–r***sid* [ **–m***mr-list* ] [ **–y** [ *comment* ] ] *s.filename* ...

## DESCRIPTION

**cdc** annotates the delta commentary for the SCCS delta ID (SID) specified by the **–r** option in each named s.file.

If the **v** flag is set in the s.file, you can also use **cdc** to update the Modification Request (MR) list.

If you checked in the delta, or, if you own the file and directory and have write permission, you can use **cdc** to annotate the commentary.

Rather than replacing the existing commentary, **cdc** inserts the new comment you supply, followed by a line of the form:

***** CHANGED ***** *yy/mm/dd hh/mm/ss username*

above the existing commentary.

If a directory is named as the *s.filename* argument, the **cdc** command applies to all s.files in that directory. Unreadable s.files produce an error; processing continues with the next file (if any). If '–' is given as the *s.filename* argument, each line of the standard input is taken as the name of an SCCS history file to be processed, and the **–m** and **–y** options must be used.

## OPTIONS

**–r***sid* Specify the SID of the delta to change.

**–m***mr-list* Specify one or more MR numbers to add or delete. When specifying more than one MR on the command line, *mr-list* takes the form of a quoted, space-separated list. To delete an MR number, precede it with a **!** character (an empty MR list has no effect). A list of deleted MRs is placed in the comment section of the delta commentary. If **–m** is not used and the standard input is a terminal, **cdc** prompts with **MRs?** for the list (before issuing the **comments?** prompt). **–m** is only useful when the **v** flag is set in the s.file. If that flag has a value, it is taken to be the name of a program to validate the MR numbers. If that validation program returns a non-zero exit status, **cdc** terminates and the delta commentary remains unchanged.

**–y**[*comment*]

Use *comment* as the annotation in the delta commentary. The previous comments are retained; the *comment* is added along with a notation that the commentary was changed. A null *comment* leaves the commentary unaffected. If **–y** is not specified and the standard input is a terminal, **cdc** prompts with **comments?** for the text of the notation to be added. An unescaped NEWLINE character terminates the annotation text.

## EXAMPLES

The following command:

**example% cdc –r1.6 –y"corrected commentary" s.program.c**

produces the following annotated commentary for delta 1.6 in **s.program.c**:

```
D 1.6 88/07/05 23:21:07 username 9 0 00001/00000/00000
MRs:
COMMENTS:
corrected commentary
*** CHANGED *** 88/07/07 14:09:41 username
performance enhancements in main()
```

**FILES**

z.*file* temporary lock file

**SEE ALSO**

**sccs**(1), **sccs-admin**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

**DIAGNOSTICS**

Use the SCCS **help** command for explanations (**sccs-help**(1)).

## NAME

sccs-comb, comb – combine SCCS deltas

## SYNOPSIS

**/usr/sccs/comb** [ **–os** ] [ **–c***sid-list* ] [ **–p***sid* ] *s.filename* ...

## DESCRIPTION

**comb** generates a shell script (see **sh**(1)) that you can use to reconstruct the indicated s.files. This script is written to the standard output.

If a directory name is used in place of the *s.filename* argument, the **comb** command applies to all s.files in that directory. Unreadable s.files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one s.file per line.

If no options are specified, **comb** preserves only the most recent (leaf) delta in a branch, and the minimal number of ancestors needed to preserve the history.

## OPTIONS

**–o** For each '**get –e**' generated, access the reconstructed file at the release of the delta to be created. Otherwise, the reconstructed file is accessed at the most recent ancestor. The use of **–o** may decrease the size of the reconstructed s.file. It may also alter the shape of the delta tree of the original file.

**–s** Generate scripts to gather statistics, rather than combining deltas. When run, the shell scripts report: the file name, size (in blocks) after combining, original size (also in blocks), and the percentage size change, computed by the formula:

100 * ( *original – combined* ) / *original*

This option can be used to calculate the space that will be saved, before actually doing the combining.

**–c***sid-list*

Include the indicated list of deltas. All other deltas are omitted. *sid-list* is a comma-separated list of SCCS delta IDs (SIDs). To specify a range of deltas, use a '–' separator instead of a comma, between two SIDs in the list.

**–p***SID* The SID of the oldest delta to be preserved.

## FILES

| | |
|---|---|
| **s.COMB** | reconstructed SCCS file |
| **comb?????** | temporary file |

## SEE ALSO

**sccs**(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-delta**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sccsdiff**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

## DIAGNOSTICS

Use the SCCS **help** command for explanations (**sccs-help**(1)).

## BUGS

**comb** may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

## NAME

sccs-delta, delta – make a delta to an SCCS file

## SYNOPSIS

**/usr/sccs/delta** [ **–nps** ] [ **–g***sid-list* ] [ **–m***mr-list* ] [ **–r***sid* ] [ **–y**[*comment*] ] *s.filename* ...

## DESCRIPTION

**delta** checks in a record of the line-by-line differences made to a checked-out version of a file under SCCS control. These changes are taken from the writable working copy that was retrieved using the SCCS **get** command (see **sccs-get**(1)). This working copy does not have the **'s.'** prefix, and is also referred to as a g-file.

If a directory name is used in place of the *s.filename*, argument, the **delta** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.file** per line (requires **–y**, and in some cases, **–m**).

**delta** may issue prompts on the standard output depending upon the options specified and the flags that are set in the **s.**file (see **sccs-admin**(1), and the **–m** and **–y** options below, for details).

## OPTIONS

**–n** Retain the edited **g**-file, which is normally removed at the completion of processing.

**–p** Display line-by-line differences (in **diff**(1) format) on the standard output.

**–s** Silent. Do not display warning or confirmation messages. Do not suppress error messages (which are written to standard error).

**–g***sid-list*
Specify a list of deltas to omit when the file is accessed at the SCCS version ID (SID) created by this delta. *sid-list* is a comma-separated list of SIDs. To specify a range of deltas, use a '–' separator instead of a comma, between two SIDs in the list.

**–m** [*mr-list*]
If the SCCS file has the **v** flag set (see **sccs-admin**(1)), you must supply one or more Modification Request (MR) numbers for the new delta. When specifying more than one MR number on the command line, *mr-list* takes the form of a quoted, space-separated list. If **–m** is not used and the standard input is a terminal, **delta** prompts with **MRs?** for the list (before issuing the **comments?** prompt). If the **v** flag in the **s.**file has a value, it is taken to be the name of a program to validate the MR numbers. If that validation program returns a non-zero exit status, **delta** terminates without checking in the changes.

**–r***sid* When two or more versions are checked out, specify the version to check in. This SID value can be either the SID specified on the **get** command line, or the SID of the new version to be checked in as reported by **get**. A diagnostic results if the specified SID is ambiguous, or if one is required but not supplied.

**–y**[*comment*]
Supply a comment for the delta table (version log). A null comment is accepted, and produces an empty commentary in the log. If **–y** is not specified and the standard input is a terminal, **delta** prompts with **'comments?'**. An unescaped NEWLINE terminates the comment.

## FILES

| | |
|---|---|
| **d.***file* | temporary file of differences |
| **p.***file* | lock file for a checked-out version |
| **q.***file* | temporary file |
| **s.***file* | SCCS history file |
| **x.***file* | temporary copy of the **s.**file |
| **z.***file* | temporary file |

**WARNINGS**

Lines beginning with an ASCII SOH character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see **sccsfile**(5)) and produces an error.

**SEE ALSO**

**sccs**(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

**DIAGNOSTICS**

Use the SCCS **help** command for explanations (**sccs-help**(1)).

## NAME

sccs-get, get – retrieve a version of an SCCS file

## SYNOPSIS

**/usr/sccs/get** [ **–begkmnpst** ] [ **–l**[ **p** ]] [ **–a***sequence* ] [ **–c***date-time* ] [ **–G***g-file* ] [ **–i***sid-list* ] [ **–r***sid* ] [ **–x***sid-list* ] *s.filename* ...

## DESCRIPTION

**get** retrieves a working copy from the SCCS history file, according to the specified options.

For each *s.filename* argument, **get** displays the SCCS delta ID (SID) and number of lines retrieved.

If a directory name is used in place of the *s.filename* argument, the **get** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

The retrieved file normally has the same filename base as the **s.**file, less the prefix, and is referred to as the **g-**file.

For each file processed, **get** responds (on the standard output) with the SID being accessed, and with the number of lines retrieved from the **s.**file.

## OPTIONS

**–b** Create a new branch. Used with the **–e** option to indicate that the new delta should have an SID in a new branch. Instead of incrementing the level for version to be checked in, **get** indicates in the **p.**file that the delta to be checked in should either initialize a new branch and sequence (if there is no existing branch at the current level), or increment the branch component of the SID. If the **b** flag is not set in the **s.**file, this option is ignored.

**–e** Retrieve a version for editing. With this option, **get** places a lock on the **s.**file, so that no one else can check in changes to the version you have checked out. If the **j** flag is set in the **s.**file, the lock is advisory: **get** issues a warning message. Concurrent use of '**get –e**' for different SIDs is allowed, however, **get** will not check out a version of the file if a writable version is present in the directory. All SCCS file protections stored in the **s.**file, including the release ceiling, floor, and authorized user list, are honored by '**get –e**'.

**–g** Get the SCCS version ID, without retrieving the version itself. Used to verify the existence of a particular SID.

**–k** Suppress expansion of ID keywords. **–k** is implied by the **–e**.

**–m** Precede each retrieved line with the SID of the delta in which it was added to the file. The SID is separated from the line with a TAB.

**–n** Precede each line with the **%M%** ID keyword and a TAB. When both the **–m** and **–n** options are used, the ID keyword precedes the SID, and the line of text.

**–p** Write the text of the retrieved version to the standard output. All messages that normally go to the standard output are written to the standard error instead.

**–s** Suppress all output normally written on the standard output. However, fatal error messages (which always go to the standard error) remain unaffected.

**–t** Retrieve the most recently created (top) delta in a given release (for example: **–r1**).

**–l**[ **p** ] Retrieve a summary of the delta table (version log) and write it to a listing file, with the '**l.**' prefix (called '**l.**file'). When **–lp** is used, write the summary onto the standard output.

**–a** *sequence*
Retrieve the version corresponding to the indicated delta sequence number. This option is used primarily by the SCCS **comb** command (see **sccs-comb**(1)); for users, **–r** is an easier way to specify a version. **–a** supercedes **–r** when both are used.

**–c***date-time*

Retrieve the latest version checked in prior to the date and time indicated by the *date-time* argument. *date-time* takes the form: *yy*[*mm*[*dd*[*hh*[*mm*[*ss*]]]]]. Units omitted from the indicated date and time default to their maximum possible values; that is **–c7502** is equivalent to **–c750228235959**. Any number of non-numeric characters may separate the various 2 digit components. If white-space characters occur, the *date-time* specification must be quoted.

**–G***newname*

Use *newname* as the name of the retrieved version.

**–i***sid-list*

Specify a list of deltas to include in the retrieved version. The included deltas are noted in the standard output message. *sid-list* is a comma-separated list of SIDs. To specify a range of deltas, use a '–' separator instead of a comma, between two SIDs in the list.

**–r***sid* Retrieve the version corresponding to the indicated SID (delta).

The SID for a given delta is a number, in Dewey decimal format, composed of two or four fields: the *release* and *level* fields, and for branch deltas, the *branch* and *sequence* fields. For instance, if **1.2** is the SID, **1** is the release, and **2** is the level number. If **1.2.3.4** is the SID, **3** is the branch and **4** is the sequence number.

You need not specify the entire SID to retrieve a version with **get**. When you omit **–r** altogether, or when you omit both release and level, **get** normally retrieves the highest release and level. If the **d** flag is set to an SID in the **s.**file and you omit the SID, **get** retrieves the default version indicated by that flag.

When you specify a release but omit the level, **get** retrieves the highest level in that release. If that release does not exist, **get** retrieves highest level from the next-highest existing release.

Similarly with branches, if you specify a release, level and branch, **get** retrieves the highest sequence in that branch.

**–x***sid-list*

Exclude the indicated deltas from the retrieved version. The excluded deltas are noted in the standard output message. *sid-list* is a comma-separated list of SIDs. To specify a range of deltas, use a '-' separator instead of a comma, between two SIDs in the list.

## USAGE

### ID Keywords

In the absence of **–e** or **–k**, **get** expands the following ID keywords by replacing them with the indicated values in the text of the retrieved source.

| *Keyword* | *Value* |
|---|---|
| **%A%** | Shorthand notation for an ID line with data for **what**(1): **%Z%%Y% %M% %I%%Z%** |
| **%B%** | SID branch component |
| **%C%** | Current line number. Intended for identifying messages output by the program such as "*this shouldn't have happened*" type errors. It is *not* intended to be used on every line to provide sequence numbers. |
| **%D%** | Current date: *yy/mm/dd* |
| **%E%** | Date newest applied delta was created: *yy/mm/dd* |
| **%F%** | SCCS **s.**file name |
| **%G%** | Date newest applied delta was created: *mm/dd/yy* |
| **%H%** | Current date: *mm/dd/yy* |
| **%I%** | SID of the retrieved version: **%R%.%L%.%B%.%S%** |
| **%L%** | SID level component |
| **%M%** | Module name: either the value of the **m** flag in the **s.**file (see **sccs-admin**(1)), or the name of the **s.**file less the prefix |
| **%P%** | Fully qualified **s.**file name |

| | |
|---|---|
| **%Q%** | Value of the **q** flag in the **s.**file |
| **%R%** | SID Release component |
| **%S%** | SID Sequence component |
| **%T%** | Current time: *hh*:*mm*:*ss* |
| **%U%** | Time the newest applied delta was created: *hh*:*mm*:*ss* |
| **%W%** | Shorthand notation for an ID line with data for **what**: **%Z%%M%** **%I%** |
| **%Y%** | Module type: value of the **t** flag in the **s.**file |
| **%Z%** | 4-character string: '@(#)', recognized by **what.** |

**FILES**

| | |
|---|---|
| "g-file" | version retrieved by **get** |
| **l.***file* | file containing extracted delta table info |
| **p.***file* | permissions (lock) file |
| **z.***file* | temporary copy of **s.***file* |

**SEE ALSO**

**sccs**(1), **sccs-admin**(1), **sccs-delta**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-sact**(1), **sccs-unget**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

**DIAGNOSTICS**

Use the SCCS **help** command for explanations (**sccs-help**(1)).

**BUGS**

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, only one file may be named when using –e.

## NAME

sccs-help, help – ask for help regarding SCCS error or warning messages

## SYNOPSIS

**/usr/sccs/help** [ *argument* ] ...

## DESCRIPTION

**help** retrieves information to further explain errors messages and warnings from SCCS commands. It also provides some information about SCCS command usage. If no arguments are given, **help** prompts for one.

An *argument* may be a message number (which normally appears in parentheses following each SCCS error or warning message), or an SCCS command name. **help** responds with an explanation of the message or a usage line for the command.

When all else fails, try '**/usr/sccs/help stuck**'.

## FILES

| | |
|---|---|
| **/usr/lib/help** | directory containing files of message text. |
| **/usr/sccs/help** | |

## SEE ALSO

**sccs**(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **sccs-val**(1), **what**(1), **sccsfile**(5)

## NAME

sccs-prs, prs – display selected portions of an SCCS history

## SYNOPSIS

**/usr/sccs/prs [ –ael ] [ –c***date-time* ] [ –d*dataspec* ] [ –r*sid* ] *s.filename* ...

## DESCRIPTION

**prs** displays part or all of the SCCS file (see **sccsfile**(5)) in a user supplied format.

If a directory name is used in place of the *s.filename* argument, the **prs** command applies to all s.files in that directory. Unreadable s.files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one s.file per line.

## OPTIONS

In the absence of options, **prs** displays the delta table (version log). In the absence of **–d**, or **–l**, **prs** displays the entry for each delta indicated by the other options.

**–a** Include all deltas, including those marked as removed (see **sccs-rmdel**(1)).

**–e** Request information for all deltas created *earlier* than, and including, the delta indicated with **–r** or **–c**.

**–l** Request information for all deltas created *later* than, and including, the delta indicated with **–r** or **–c**.

**–c***date-time*

Display information on the latest delta checked in prior to the date and time indicated by the *date-time* argument. *date-time* takes the form:

*yy*[*mm*[*dd*[*hh*[*mm*[*ss*]]]]]

Units omitted from the indicated date and time default to their maximum possible values; that is **–c7502** is equivalent to **–c750228235959**. Any number of non-numeric characters may separate the various 2 digit components. If white-space characters occur, the *date-time* specification must be quoted.

**–d***dataspec*

Produce a report according to the indicated data specification. *dataspec* consists of a (quoted) text string that includes embedded data keywords of the form: '*:key:*' (see *Data Keywords*, below). **prs** expands these keywords in the output it produces. To specify a TAB character in the output, use \t; to specify a NEWLINE in the output, use \n.

**–r***sid* Specify the SCCS delta ID (SID) of the delta for which information is desired. If no SID is specified, the most recently created delta is used.

## USAGE

### Data Keywords

Data keywords specify which parts of an SCCS file are to be retrieved. All parts of an SCCS file (see **sccsfile**(5)) have an associated data keyword. A data keyword may appear any number of times in a data specification argument to **–d**. These data keywords are listed in the table below:

| *Keyword* | *Data Item* | *File Section** | *Value* | *Format*** |
|---|---|---|---|---|
| **:A:** | a format for the **what** string: | N/A | **:Z::Y: :M: :I::Z:** | S |
| **:B:** | branch number | D | *nnnn* | S |
| **:BD:** | body | B | *text* | M |
| **:BF:** | branch flag | F | **yes** or **no** | S |
| **:CB:** | ceiling boundary | F | **:R:** | S |
| **:C:** | comments for delta | D | *text* | M |
| **:D:** | date delta created | D | **:Dy:/:Dm:/:Dd:** | S |
| **:Dd:** | day delta created | D | *nn* | S |

| | | | | |
|---|---|---|---|---|
| **:Dg:** | deltas ignored (seq #) | D | **:DS: :DS:** ... | S |
| **:DI:** | seq-no. of deltas included, excluded, ignored | D | **:Dn:/:Dx:/:Dg:** | S |
| **:DL:** | delta line statistics | D | **:Li:/:Ld:/:Lu:** | S |
| **:Dm:** | month delta created | D | *nn* | S |
| **:Dn:** | deltas included (seq #) | D | **:DS: :DS:** ... | S |
| **:DP:** | predecessor delta seq-no. | D | *nnnn* | S |
| **:Ds:** | default SID | F | **:I:** | S |
| **:DS:** | delta sequence number | D | *nnnn* | S |
| **:Dt:** | delta information | D | **:DT: :I: :D: :T: :P: :DS: :DP:** | S |
| **:DT:** | delta type | D | **D** or **R** | S |
| **:Dx:** | deltas excluded (seq #) | D | **:DS:** ... | S |
| **:Dy:** | year delta created | D | *nn* | S |
| **:F:** | **s.**file name | N/A | *text* | S |
| **:FB:** | floor boundary | F | **:R:** | S |
| **:FD:** | file descriptive text | C | *text* | M |
| **:FL:** | flag list | F | *text* | M |
| **:GB:** | gotten body | B | *text* | M |
| **:I:** | SCCS delta ID (SID) | D | **:R:.:L:.:B:.:S:** | S |
| **:J:** | joint edit flag | F | **yes** or **no** | S |
| **:KF:** | keyword error/warning flag | F | **yes** or **no** | S |
| **:L:** | level number | D | *nnnn* | S |
| **:Ld:** | lines deleted by delta | D | *nnnnn* | S |
| **:Li:** | lines inserted by delta | D | *nnnnn* | S |
| **:LK:** | locked releases | F | **:R:** ... | S |
| **:Lu:** | lines unchanged by delta | D | *nnnnn* | S |
| **:M:** | module name | F | *text* | S |
| **:MF:** | MR validation flag | F | **yes** or **no** | S |
| **:MP:** | MR validation program | F | *text* | S |
| **:MR:** | MR numbers for delta | D | *text* | M |
| **:ND:** | null delta flag | F | **yes** or **no** | S |
| **:Q:** | user defined keyword | F | *text* | S |
| **:P:** | user who created delta | D | *username* | S |
| **:PN:** | **s.**file's pathname | N/A | *text* | S |
| **:R:** | release number | D | *nnnn* | S |
| **:S:** | sequence number | D | *nnnn* | S |
| **:T:** | time delta created | D | **:Th:::Tm:::Ts:** | S |
| **:Th:** | hour delta created | D | *nn* | S |
| **:Tm:** | minutes delta created | D | *nn* | S |
| **:Ts:** | seconds delta created | D | *nn* | S |
| **:UN:** | user names | U | *text* | M |
| **:W:** | a form of **what** string | N/A | **:Z::M:\t:I:** | S |
| **:Y:** | module type flag | F | *text* | S |
| **:Z:** | **what** string delimiter | N/A | **@(#)** | S |

*B = body, D = delta table, F = flags, U = user names

**S = simple format, M = multi-line format

**EXAMPLES**

The command:

**/usr/sccs/prs –e –d":I:\t:P:" program.c**

produces:

**1.6 username**
**1.5 username**
...

**FILES**

**/tmp/pr?????** temporary file

**SEE ALSO**

**sccs**(1), **sccs-cdc**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prt**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

**DIAGNOSTICS**

Use the SCCS **help** command for explanations (**sccs-help**(1)).

## NAME

sccs-prt, prt – display delta table information from an SCCS file

## SYNOPSIS

**/usr/sccs/prt** [**–abdefistu**] [ –c*date-time* ] [ –r*date-time* ] [ –y*sid* ] *s.filename* ...

## DESCRIPTION

**prt** prints selected portions of an SCCS file. By default, it prints the delta table (version log).

If a directory name is used in place of the *s.filename* argument, the **prt** command applies to all s.files in that directory. Unreadable s.files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one s.file per line.

## OPTIONS

If any option other than **–y**, **–c**, or **–r** is supplied, the name of each file being processed (preceded by one NEWLINE and followed by two NEWLINE characters) appears above its contents.

If none of the **–u**, **–f**, **–t**, or **–b** options are used, **–d** is assumed. **–s**, **–i** are mutually exclusive, as are **–c** and **–r**.

**–a** Display log entries for all deltas, including those marked as removed.

**–b** Print the body of the s.file.

**–d** Print delta table entries. This is the default.

**–e** Everything. This option implies **–d**, **–i**, **–u**, **–f**, and **–t**.

**–f** Print the flags of each named s.file.

**–i** Print the serial numbers of included, excluded, and ignored deltas.

**–s** Print only the first line of the delta table entries; that is, only up to the statistics.

**–t** Print the descriptive text contained in the s.file.

**–u** Print the user-names and/or numerical group IDs of users allowed to make deltas.

**–c***date-time*

Exclude delta table entries that are specified cutoff date and time. Each entry is printed as a single line, preceded by the name of the SCCS file. This format (also produced by **–r** , and **–y**) makes it easy to sort multiple delta tables in chronological order. When both **–y** and **–c**, or **–y** and **–r** are supplied, **prt** stops printing when the first of the two conditions is met.

**–r***date-time*

Exclude delta table entries that are newer than the specified cutoff date and time.

**–y***sid* Exclude delta table entries made prior to the SID specified. If no delta in the table has the specified SID, the entire table is printed. If no SID is specified, the most recent delta is printed.

## USAGE

### Output Format

The following format is used to print those portions of the s.file that are specified by the various options.

- NEWLINE
- Type of delta (D or R)
- SPACE
- SCCS delta ID (SID)
- TAB
- Date and time of creation in the form: *yy/mm/dd hh/mm/ss*

- SPACE
- Username the delta's creator
- TAB
- Serial number of the delta
- SPACE
- Predecessor delta's serial number
- TAB
- Line-by-line change statistics in the form: *inserted/deleted/unchanged*
- NEWLINE
- List of included deltas, followed by a NEWLINE (only if there were any such deltas and the **–i** options was used)
- List of excluded deltas, followed by a NEWLINE (only if there were any such deltas and the **–i** options was used)
- List of ignored deltas, followed by a NEWLINE (only if there were any such deltas and the **–i** options was used)
- List of modification requests (MR s), followed by a NEWLINE (only if any MR numbers were supplied).
- Lines of the delta commentary (if any), followed by a NEWLINE.

**EXAMPLES**

The command:

**/usr/sccs/prt –y program.c**

produces a one-line display of the delta table entry for the most recent version:

**s.program.c: D 1.6 88/07/06 21:39:39 username 5 4 00159/00080/00636 ...**

**SEE ALSO**

**sccs**(1), **sccs-cdc**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

**DIAGNOSTICS**

Use the SCCS **help** command for explanations (**sccs-help**(1)).

## NAME

sccs-rmdel, rmdel – remove a delta from an SCCS file

## SYNOPSIS

**/usr/sccs/rmdel** –r*sid s.filename* ...

## DESCRIPTION

**rmdel** removes the delta specified by the SCCS delta ID (SID) supplied with **–r**. The delta to be removed must be the most recent (leaf) delta in its branch. In addition, the SID must *not* be that of a version checked out for editing: it must not appear in any entry of the version lock file (**p.**file).

If you created the delta, or, if you own the file and directory and have write permission, you can remove it with **rmdel**.

If a directory name is used in place of the *s.filename* argument, the **rmdel** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

## FILES

| | |
|---|---|
| **s.***file* | history file |
| **z.***file* | temporary copy of the **s.**file |
| **p.***file* | permissions file |

## SEE ALSO

**sccs**(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

## DIAGNOSTICS

Use the SCCS **help** command for explanations (**sccs-help**(1)).

## NAME

sccs-sact, sact – show editing activity status of an SCCS file

## SYNOPSIS

**/usr/sccs/sact** *s.filename* ...

## DESCRIPTION

**sact** informs the user of any SCCS files that are checked out for editing.

The output for each named file consists of five fields separated by SPACE characters.

- SID of a delta that currently exists in the SCCS file, to which changes will be made to make the new delta
- SID for the new delta to be created
- Username of the person who has the file checked out for editing.
- Date that the version was checked out.
- Time that the version was checked out.

If a directory name is used in place of the *s.filename* argument, the **sact** command applies to all s.files in that directory. Unreadable s.files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one s.file per line.

## SEE ALSO

**sccs**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

## DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

## BUGS

**sact** is not recognized as a subcommand of **sccs**(1).

## NAME

sccs-sccsdiff, sccsdiff – compare two versions of an SCCS file

## SYNOPSIS

**/usr/sccs/sccsdiff** [ **–p** ] –r*sid* –r*sid* [ *diff-options* ] *s.filename*

## DESCRIPTION

**sccsdiff** compares two versions of an SCCS file and displays the differences between the two versions. Any number of SCCS files may be specified; the options specified apply to all named **s.**files.

## OPTIONS

**–p** Pipe output for each file through **pr**(1V).

–r*sid* Specify a version corresponding to the indicated SCCS delta ID (SID) for comparison. Versions are passed to **diff**(1) in the order given.

*diff-options*

Pass options to **diff**(1), including: **–c**, **–e**, **–f**, **–h**, **–b** and **–D**.

## FILES

**/tmp/get?????** temporary files

## SEE ALSO

**diff**(1), **sccs**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

## DIAGNOSTICS

*filename*: **No differences**

If the two versions are the same.

Use the SCCS **help** command for explanations of other messages (see **sccs-help**(1)).

## NAME

sccs-unget, unget – undo a previous get of an SCCS file

## SYNOPSIS

**/usr/sccs/unget** [ **–ns** ] [ **–r***sid* ] *s.filename* ...

## DESCRIPTION

**unget** undoes the effect of a 'get –e' done prior to the creation of the pending delta.

If a directory name is used in place of the *s.filename* argument, the **unget** command applies to all s.files in that directory. Unreadable s.files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one s.file per line.

## OPTIONS

**–n** Retain the retrieved version, which is otherwise removed.

**–s** Suppress display of the SCCS delta ID (SID).

**–r***sid* When multiple versions are checked out, specify which pending delta to abort. A diagnostic results if the specified SID is ambiguous, or if it is necessary but omitted from the command line.

## SEE ALSO

**sccs**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **what**(1), **sccsfile**(5)

## DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

## NAME

sccs-val, val – validate an SCCS file

## SYNOPSIS

**/usr/sccs/val –**

**/usr/sccs/val** [ **–s** ] [ **–m** *name* ] [ **–r***sid* ] [ **–y** *type* ] *s.filename ...*

## DESCRIPTION

**val** determines if the specified s.files files meet the characteristics specified by the indicated arguments. **val** can process up to 50 files on a single command line.

**val** has a special argument, '–', which reads the standard input until the end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

**val** generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8–bit code upon exit as described below.

The 8-bit code returned by **val** is a disjunction of the possible errors, that is, it can be interpreted as a bit string where (moving from left to right) the bits set are interpreted as follows:

bit 0 = missing file argument
bit 1 = unknown or duplicate option
bit 2 = corrupted s.file
bit 3 = can not open file or file not in s.file format
bit 4 = the SCCS delta ID (SID) is invalid or ambiguous
bit 5 = the SID does not exist
bit 6 = mismatch between **%Y%** and **–y** argument
bit 7 = mismatch between **%M%** **–m** argument

**val** can process two or more files on a given command line, and in turn can process multiple command lines (when reading the standard input). In these cases, an aggregate code is returned which is the logical **OR** of the codes generated for each command line and file processed.

## OPTIONS

**–s** Silent. Suppress the normal error or warning messages.

**–m***name* Compare *name* with the **%M%** ID keyword in the s.file.

**–r***sid* Check to see if the indicated SID is ambiguous, invalid, or absent from the s.file.

**–y** *type* Compare *type* with the **%Y%** ID keyword.

## SEE ALSO

**sccs**(1), **sccs-admin**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

## DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

## NAME

screenblank – turn off the screen when the mouse and keyboard are idle

## SYNOPSIS

**screenblank** [ **–m** ] [ **–k** ] [ **–d** *interval* ] [ **–e** *interval* ] [ **–f** *frame-buffer* ]

## DESCRIPTION

**screenblank** turns off the display when the mouse and keyboard are idle for an extended period (the default is 10 minutes). **screenblank** will continue to run until killed by hand, using '**kill** *processid*'.

Place the **screenblank** command in your **/etc/rc.local** file.

## OPTIONS

**–m** Do not check whether the mouse has been idle.

**–k** Do not check whether the keyboard has been idle.

**–d** *interval*

Disable after *interval* seconds. *interval* is a number of the form *xxx.xxx* where each *x* is a decimal digit. The default is 600 seconds (10 minutes).

**–e** *interval*

Enable within *interval* seconds. *interval* is the time between successive polls for keyboard or mouse activity. If a poll detects keyboard or mouse activity, the display is resumed. *interval* is a number of seconds, of the form *xxx.xxx* where each *x* is a decimal digit. The default is 0.25 seconds.

**–f** *frame-buffer*

*frame-buffer* is the path name of the frame-buffer on which video disabling/enabling applies. The defaults is **/dev/fb**.

## FILES

**/dev/fb**

## SEE ALSO

**lockscreen**(1), **sunview**(1)

## BUGS

**screenblank** only checks **/dev/console** for activity; it does not check non-window programs from **/dev/tty** which bypass **/dev/console** . Consequently, **screenblank** will turn off the display if **/dev/tty** programs (for example, pixrect-based programs) are the only ones running.

When not running **sunview**(1), only the RETURN key resumes video display.

**NAME**

screendump – dump a frame-buffer image to a file

**SYNOPSIS**

**screendump** [ **–cCeo** ] [ **–f** *frame-buffer* ] [ **–t** *type* ] [ **–xyXY** *value* ] [ *filename* ]

**DESCRIPTION**

**screendump** reads the contents of a frame buffer and writes the display image to *filename* (the default is the standard output ) in Sun standard rasterfile format.

If the frame buffer has both an overlay plane and color planes, **screendump** examines the overlay enable plane and tries to make the output file represent what is visible on the screen. It maps the overlay plane foreground and background colors into the closest values present in the color map for the color planes.

**OPTIONS**

| | |
|---|---|
| **–c** | Dump the frame-buffer contents directly without making a temporary copy in a memory pixrect. Saves time and memory but lengthens the time the frame-buffer must be inactive to guarantee a consistent screen dump. |
| **–C** | Dump the frame-buffer color planes only (ignored if the display does not have color planes). |
| **–e** | Set the output rasterfile type to 2, RT_BYTE_ENCODED. For most images this saves a significant amount of space compared to the standard format. |
| **–o** | Dump the frame-buffer overlay plane only (ignored if the display does not have an overlay plane). |
| **–f** *frame-buffer* | Dump the specified frame-buffer device (default is **/dev/fb**). |
| **–t** *type* | Set the output rasterfile type (default 1, RT_STANDARD). |
| **–x** *value*<br>**–y** *value* | Set the x or y coordinate of the upper left corner of the area to be dumped to the given value. |
| **–X** *value*<br>**–Y** *value* | Set the width or height of the area to be dumped to the given value. |

**EXAMPLES**

The command:

**example% screendump save.this.image**

writes the current contents of the console frame buffer into the file **save.this.image**,

while the command:

**example% screendump –f /dev/cgtwo save.color.image**

writes the current contents of the color frame buffer **/dev/cgtwo** into the file **save.color.image**.

The command:

**example% screendump | lpr –Pversatec –v**

sends a rasterfile containing the current frame-buffer to the lineprinter, selecting the printer **versatec** and the **v** output filter (see **/etc/printcap**). The printer must support an appropriate imaging model such as PostScript in order to print the image.

**FILES**

| | |
|---|---|
| **/usr/lib/rasfilters/*** | filters for non-standard rasterfile formats |
| **/dev/fb** | default frame buffer device |
| **/etc/printcap** | |

**SEE ALSO**

**lpr**(1), **rasfilter8to1**(1), **rastrepl**(1), **screenload**(1)

*Pixrect Reference Manual*

**BUGS**

The output file or the screen may be corrupted if the frame-buffer contents are modified while the dump is in progress.

## NAME

screenload – load a frame-buffer image from a file

## SYNOPSIS

**screenload** [ **–bgnw** ] [ **–dopr** ] [ **–f***frame-buffer* ] [ **–h** *count data ...* ] [ **–i** *color* ] [ **–xyXY** *value* ] [ *filename* ]

## DESCRIPTION

**screenload** reads a Sun standard rasterfile (see **rasterfile**(5)) and displays it on a frame-buffer. **screenload** is able to display monochrome images on a color display, but cannot display color images on a monochrome display. If the input file contains a color image, a frame-buffer has not been explicitly specified, and **/dev/fb** is a monochrome frame-buffer, **screenload** looks for a color frame-buffer with one of the standard device names.

If the image contained in the input file is larger than the actual resolution of the display, **screenload** clips the right and bottom edges of the image. If the input image is smaller than the display (for example, loading an 1152-by-900 image on a 1600-by-1280 high resolution display), **screenload** centers the image on the display surface and fills the border area with solid black (by default). Various options may be used to change the image location, or to change or disable the fill pattern.

## OPTIONS

**–b** Fill the border area with a pattern of solid ones (default). On a monochrome display this results in a black border; on a color display the color map value selected by the **–i** option determines the border color.

**–g** Fill the border area with a pattern of "desktop grey". On a monochrome display this results in a border matching the default background pattern used by SunView; on a color display the color map value selected by the **–i** option determines the foreground border color, though the pattern is the same as on a monochrome display.

**–n** Do not fill the border area.

**–w** Fill the border area with a pattern of solid zeros. On a monochrome display this results in a white border; on a color display the color map value at index 0 determines the border color.

**–d** Print a warning message if the display size does not match the rasterfile image size.

**–o** Load the image on the overlay plane of the display (ignored if the display does not have an overlay plane).

**–p** Wait for a NEWLINE to be typed on the standard input before exiting.

**–r** Reverse the foreground and background of the output image. Useful when loading a screendump made from a reverse video screen.

**–f***frame-buffer* Display the image on the specified frame-buffer device (default **/dev/fb**).

**–h** *count data ...* Fill the border area with the bit pattern described by the following *count* 16-bit hexadecimal constants. Note: a "1" bit is black and a "0" bit is white on the monochrome display; on a color display the color map value selected by the **–i** option determines the border foreground color. The number of hex constants in the pattern is limited to 16.

**–i** *color* Fill the border area with the given color value (default 255).

**–x** *value*
**–y** *value* Set the x or y coordinate of the upper left corner of the image on the display to the given value.

**-X** *value*
**-Y** *value* Set the maximum width or height of the displayed image to the given value.

**EXAMPLES**

The command:

**example% screenload saved.display.image**

loads the raster image contained in the file **saved.display.image** on the display type indicated by the rasterfile header in that file.

The command:

**example% screenload -f /dev/cgtwo monochrome.image**

reloads the raster image in the file **monochrome.image** on the color frame-buffer device **/dev/cgtwo.**

The command:

**example% screenload -h1 ffff small.saved.image**

is equivalent to the **-b** option (fill border with black), while

**example% screenload -h4 8888 8888 2222 2222 small.saved.image**

is equivalent to the **-g** option (fill border with desktop grey).

**FILES**

| | |
|---|---|
| **/usr/lib/rasfilters/*** | filters for non-standard rasterfile formats |
| **/dev/fb** | default frame buffer device |

**SEE ALSO**

**rasfilter8to1**(1), **rastrepl**(1), **screendump**(1), **screenload**(1)

*Pixrect Reference Manual*

## NAME

script – make typescript of a terminal session

## SYNOPSIS

**script** [ **–a** ] [ *filename* ]

## DESCRIPTION

**script** makes a typescript of everything printed on your terminal. The typescript is written to *filename*, or appended to *filename* if the **–a** option is given. It can be sent to the line printer later with **lpr**(1). If no file name is given, the typescript is saved in the file **typescript**.

The script ends when the forked shell exits.

## OPTIONS

**–a** Append the script to the specified file instead of writing over it.

## SEE ALSO

**lpr**(1)

## BUGS

**script** places *everything* in the log file. This is not what the naive user expects.

## NAME

sdiff – contrast two text files by displaying them side-by-side

## SYNOPSIS

**sdiff** [ **–l** ] [ **–o** *outfile* ] [ **–s** ] [ **–w** *n* ] *filename1 filename2*

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**sdiff** uses the output of **diff**-b to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *filename1*, a > in the gutter if the line only exists in *filename2*, and a | for lines that are different. See EXAMPLES.

## OPTIONS

**–w** *n* Use *n* as the width of the output line. The default line length is 130 characters.

**–l** Only print the left side of any identical lines.

**–s** Silent. Do not print identical lines.

**–o** *outfile*

Use the next argument, *output*, as the name of an output file created as an interactively controlled merging of *filename1* and *filename2*. Identical lines of *filename1* and *filename2* are copied to *output*. Sets of differences, as produced by **diff**, are printed; where a set of differences share a common gutter character. After printing each set of differences, **sdiff** prompts with a % and waits for you to type one of the following commands:

**l** Append the left column to the output file.

**r** Append the right column to the output file.

**s** Turn on silent mode; do not print identical lines.

**v** Turn off silent mode.

**e l** Call the **ed**(1) with the left column.

**e r** Call **ed**(1) with the right column.

**e b** Call **ed**(1) with the concatenation of left and right columns.

**e** Call **ed**(1) with a zero length file.

On exit from **ed**(1), the resulting file is concatenated to the named output file.

**q** Exit from the program.

## EXAMPLES

A sample output of **sdiff** would look like this:

```
x     |     y
a           a
b     <
c     <
d           d
      >     c
```

**SEE ALSO**

**diff**(1), **ed**(1)

## NAME

sed – stream editor

## SYNOPSIS

**sed** [ **–n** ] [ **–e** *script* ] [ **–f** *sfilename* ] [ *filename* ]...

## SYSTEM V SYNOPSIS

**/usr/5bin/sed** [ **–n** ] [ **–e** *script* ] [ **–f** *sfilename* ] [ *filename* ]...

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**sed** copies the *filename*s (standard input default) to the standard output, edited according to a script of commands.

## OPTIONS

**–n** Suppress the default output.

**–e** *script* *script* is an edit command for **sed**. If there is just one –e option and no –f options, the **–e** flag may be omitted.

**–f** *sfilename* Take the script from *sfilename*.

## USAGE

### sed Scripts

**sed scripts** consist of editing commands, one per line, of the following form:

[ *address* [**,** *address* ] ] *function* [ *arguments* ]

In normal operation **sed** cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), sequentially applies all commands with *addresses* matching that pattern space until reaching the end of the script, copies the pattern space to the standard output (except under **–n**), and finally, deletes the pattern space.

Some commands use a *hold space* to save all or part of the pattern space for subsequent retrieval.

An *address* is either:

a decimal number linecount, which is cumulative across input files;

a $, which addresses the last input line;

or a context address, which is a */regular expression/* in the style of **ed**(1);

with the following exceptions:

\?RE? In a context address, the construction \ *?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note: in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef.**

**\n** Matches a NEWLINE embedded in the pattern space.

**.** Matches any character except the NEWLINE ending the pattern space.

*null* A command line with no address selects every pattern space.

*address* Selects each pattern space that matches.

*address1* ,*address2*

Selects the inclusive range from the first pattern space matching *addrress1* to the first pattern space matching *address2*. Selects only one line if *address1* is greater than or equal to *address2*.

**Comments**

If the first nonwhite character in a line is a '#' (pound sign), **sed** treats that line as a comment, and ignores it. If, however, the first such line is of the form:

**#n**

**sed** runs as if the **–n** flag were specified.

**Functions**

The maximum number of permissible addresses for each function is indicated in parentheses in the list below.

An argument denoted *text* consists of one or more lines, all but the last of which end with \ to hide the NEWLINE. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial SPACE and TAB characters against the stripping that is done on every script line.

An argument denoted *rfilename* or *wfilename* must terminate the command line and must be preceded by exactly one SPACE. Each *wfilename* is created before processing begins. There can be at most 10 distinct *wfilename* arguments.

(1)**a**
*text* Append: place *text* on the output before reading the next input line.

(2) **b** *label* Branch to the ':' command bearing the *label*. Branch to the end of the script if *label* is empty.

(2) **c**
*text* Change: delete the pattern space. With 0 or 1 address or at the end of a 2 address range, place *text* on the output. Start the next cycle.

(2) **d** Delete the pattern space. Start the next cycle.

(2) **D** Delete the initial segment of the pattern space through the first NEWLINE. Start the next cycle.

(2) **g** Replace the contents of the pattern space by the contents of the hold space.

(2) **G** Append the contents of the hold space to the pattern space.

(2) **h** Replace the contents of the hold space by the contents of the pattern space.

(2) **H** Append the contents of the pattern space to the hold space.

(1) **i**
*text* Insert: place *text* on the standard output.

(2) **l** List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two digit ASCII and long lines are folded.

(2) **n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) **N** Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2) **p** Print: copy the pattern space to the standard output.

(2) **P** Copy the initial segment of the pattern space through the first NEWLINE to the standard output.

(1) **q** Quit: branch to the end of the script. Do not start a new cycle.

(2) **r** *rfilename*
Read the contents of *rfilename*. Place them on the output before reading the next input line.

(2) **s**/*regular expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of '/'. For a fuller description see **ed**(1). *flags* is zero or more of:

*n* *n*= 1 – 512. Substitute for just the *n*th occurrence of the *regular*expression.

**g** Global: substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

**p** Print the pattern space if a replacement was made.

**w** *wfilename* Write: append the pattern space to *wfilename* if a replacement was made.

(2) **t** *label* Test: branch to the '**:**' command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.

(2) **w** *wfilename*
Write: append the pattern space to *wfilename*.

(2) **x** Exchange the contents of the pattern and hold spaces.

(2) **y**/*string1*/*string2*/
Transform: replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)**!** *function* Do not: apply the *function* (or group, if *function* is '{') only to lines *not* selected by the address(es).

(0) **:** *label* This command does nothing; it bears a *label* for **b** and **t** commands to branch to. Note: the maximum length of *label* is seven characters.

(1) **=** Place the current line number on the standard output as a line.

(2) **{** Execute the following commands through a matching '}' only when the pattern space is selected. Commands are separated by '**;**'.

(0) An empty command is ignored.

**System V sed Scripts**

Initial SPACE and TAB characters are *not* stripped from text lines.

## DIAGNOSTICS

**Too many commands**

The command list contained more than 200 commands.

**Too much command text**

The command list was too big for **sed** to handle. Text in the **a**, **c**, and **i** commands, text read in by **r** commands, addresses, regular expressions and replacement strings in **s** commands, and translation tables in **y** commands all require **sed** to store data internally.

**Command line too long**

A command line was longer than 4000 characters.

**Too many line numbers**
More than 256 decimal number linecounts were specified as addresses in the command list.

**Too many files in w commands**
More than 10 different files were specified in **w** commands or **w** options for **s** commands in the command list.

**Too many labels**
More than 50 labels were specified in the command list.

**Unrecognized command**
A command was not one of the ones recognized by **sed**.

**Extra text at end of command**
A command had extra text after the end.

**Illegal line number**
An address was neither a decimal number linecount, a **$**, nor a context address.

**Space missing before filename**
There was no space between a **r** or **w** command, or the **w** option for a **s** command, and the filename specified for that command.

**Too many {'s**
There were more { than } in the list of commands to be executed.

**Too many }'s**
There were more } than { in the list of commands to be executed.

**No addresses allowed**
A command that takes no addresses had an address specified.

**Only one address allowed**
A command that takes one address had two addresses specified.

**"\digit" out of range**
The number in a *n* item in a regular expression or a replacement string in a **s** command was greater than 9.

**Bad number**
One of the endpoints in a range item in a regular expression (that is, an item of the form {*n*} or {*n*,*m*}) was not a number.

**Range endpoint too large**
One of the endpoints in a range item in a regular expression was greater than 255.

**More than 2 numbers given in \{ \}**
More than two endpoints were given in a range expression.

**} expected after **
A \ appeared in a range expression and was not followed by a }.

**First number exceeds second in \{ \}**
The first endpoint in a range expression was greater than the second.

**Illegal or missing delimiter**
The delimiter at the end of a regular expression was absent.

**\( \) imbalance**
There were more \( than \), or more \) than \(, in a regular expression.

**[ ] imbalance**
There were more [ than ], or more ] than [, in a regular expression.

**First RE may not be null**
The first regular expression in an address or in a **s** command was null (empty).

**Ending delimiter missing on substitution**
The ending delimiter in a **s** command was absent.

**Ending delimiter missing on string**
The ending delimiter in a **y** command was absent.

**Transform strings not the same size**
The two strings in a **y** command were not the same size.

**Suffix too large - 512 max**
The suffix in a **s** command, specifying which occurrence of the regular expression should be replaced, was greater than 512.

**Label too long**
A label in a command was longer than 8 characters.

**Duplicate labels**
The same label was specified by more than one **:** command.

**File name too long**
The filename specified in a **r** or **w** command, or in the **w** option for a **s** command, was longer than 1024 characters.

**Output line too long.**
An output line was longer than 4000 characters long.

**Too many appends or reads after line *n***
More than 20 **a** or **r** commands were to be executed for line *n*.

**Hold space overflowed.**
More than 4000 characters were to be stored in the *hold space*.

## SEE ALSO

**awk**(1), **ed**(1), **grep**(1V), **lex**(1)

*Editing Text Files*

## BUGS

There is a combined limit of 200 **–e** and **–f** arguments. In addition, there are various internal size limits which, in rare cases, may overflow. To overcome these limitations, either combine or break out scripts, or use a pipeline of **sed** commands.

## NAME

selection_svc – SunView selection service

## SYNOPSIS

**selection_svc** [ **–d** ]

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

The SunView selection service handles the state and rank of various selections for its client programs. A selection service is started automatically by **sunview**(1). However, you can also start one explicitly using the **selection_svc** command.

## OPTIONS

**–d** Debug a client program. Use the alternate socket provided for testing and debugging a client program. The second selection service invoked with the **–d** handles a different set of selections, and can co-exist with one started in the normal fashion. Each service responds only to requests directed to its own socket. The client to be debugged can be directed to use the "debugging" service by using the **seln_use_test_service()** procedure, as described in the *SunView System Programmer's Guide*.

## SEE ALSO

**sunview**(1)

*SunView User's Guide*

*SunView System Programmer's Guide*

## NAME

set_alarm, get_alarm, ring_alarm – SunView programmable alarms

## SYNOPSIS

**set_alarm** [ **–b** *number* ] [ **–f** *number* ] [ **–d** *duration* ]

**get_alarm**

**ring_alarm**

## AVAILABILITY

This command is available with the *SunView User's* software installation option. Refer to *Installing SunOS 4.1* for more information on how to install optional software.

## DESCRIPTION

**set_alarm** sets the **WINDOW_ALARM** environment variable according to the options specified.

**get_alarm** returns the setting of **WINDOW_ALARM** in the form:

**WINDOW_ALARM=***setting*

**ring_alarm** performs the actual beeping and flashing of the alarm. Note: the occurrence of the alarm is controlled by the **SunView/Visible_Bell** and **SunView/Audible_Bell** settings in **defaultsedit**(1). If either of these is disabled, the respective feature of the alarm will not occur, regardless of how it is specified by **WINDOW_ALARM.**

In the C shell, **set_alarm** echoes the following command to set the **WINDOW_ALARM** environment variable:

**set noglob;setenv WINDOW_ALARM** *setting***;unset noglob;**

In the Bourne shell, **set_alarm** echos the following:

**export WINDOW_ALARM;WINDOW_ALARM=***setting***;**

Consequently, **set_alarm** is should be used as follows:

**eval 'set_alarm** [*options*]'

## OPTIONS

| | |
|---|---|
| **–b** *number* | Specify the number of beeps. The default is 1. |
| **–f** *number* | Specify the number of flashes. The default is 1. |
| **–d** *duration* | Specify the duration of the beep in milliseconds. The default is 1000. |

## ENVIRONMENT

The SunView programmable alarms use the **WINDOW_ALARM** environment variable to define the characteristics of the alarm. The variable is set with a string argument of the following form:

**:beeps=***b***:flashes=***f***:dur=***t***:**

## SEE ALSO

**csh**(1), **defaultsedit**(1), **sh**(1)

*SunView User's Guide*

## DIAGNOSTICS

**get_alarm: the WINDOW_ALARM environment variable is NULL.**
This message is self-explanatory.

**ring_alarm: WINDOW_ALARM is NULL; using default values.**
**set_alarm** was not used to set values for the WINDOW_ALARM environment variable. See OPTIONS for default values.

**ring_alarm: WINDOW_ME not found**
Attempted to ring the alarm when SunView is not running.

**BUGS and LIMITATIONS**

The user must be running SunView in order to ring the alarm; consequently, the alarms cannot be used with Open Windows.

## NAME

sh – shell, the standard UNIX system command interpreter and command-level language

## SYNOPSIS

**sh** [ **–acefhiknstuvx** ] [ *arguments* ]

## DESCRIPTION

**sh**, the Bourne shell, is the standard UNIX-system command interpreter. It executes commands read from a terminal or a file.

### Definitions

A *blank* is a TAB or a SPACE character. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *, @, #, ?, –, $, and ! .

### Invocation

If the shell is invoked through **execve**(2V), **exec( )**, see **execl**(3V), and the first character of argument zero is '–', commands are initially read from **/etc/profile** and from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **sh**.

## OPTIONS

The options below are interpreted by the shell on invocation only; unless the **–c** or **–s** option is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters for use with the commands that file contains.

**–i** If the **–i** option is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that '**kill 0**' does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

**–s** If the **–s** option is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for **Special Commands**) is written to file descriptor 2.

**–c** *string* If the **–c** option is present commands are read from *string*.

The remaining options and arguments are described under the **set** command, under **Special Commands**, below.

## USAGE

Refer to *SunOS User's Guide: Doing More* for more information about using the shell as a programming language.

### Commands

A *simple command* is a sequence of nonblank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see **execve**(2V)). The *value* of a *simple command* is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see **sigvec**(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by '|' (or, for historical compatibility, by '^'). The standard output of each command but the last is connected by a **pipe** (see **pipe**(2V)) to the standard input of the next command. Each command is run as a separate process; the shell normally waits for the last command to terminate before prompting for or accepting the next input line. The exit status of a pipeline is the exit status of its last command.

A *list* is a sequence of one or more *simple command*s or pipelines, separated by ';', '&', '&&', or '||', and optionally terminated by ';' or '&'. Of these four symbols, ';' and '&' have equal precedence, which is lower than that of '&&' and '||'. The symbols '&&' and '||' also have equal precedence. A semicolon (;) sequentially executes the preceding pipeline; an ampersand (&) asynchronously executes the preceding pipeline (the shell does *not* wait for that pipeline to finish). The symbols && and || are used to indicate conditional execution of the list that follows. With && , *list* is executed only if the preceding pipeline (or command) returns a zero exit status. With ||, *list* is executed only if the preceding pipeline (or command) returns a nonzero exit status. An arbitrary number of NEWLINE characters may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a *simple command* or one of the following constructions. Unless otherwise stated, the value returned by a command is that of the last *simple command* executed in the construction.

**for** *name* [ **in** *word* ... ] **do** *list* **done**
Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see **Parameter Substitution** below). Execution ends when there are no more words in the list.

**case** *word* **in** [*pattern*[ | *pattern* ] ... ) *list* ;; ] ... **esac**
A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see **Filename Generation**) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

**if** *list* **then** *list* [ **elif** *list* **then** *list* ] ... [ **else** *list* ] **fi**
The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

**while** *list* **do** *list* **done**
A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*) Execute *list* in a subshell.

{*list*;} *list* is simply executed.

*name* () {*list*;}
Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see **Execution**).

The following words are only recognized as the first word of a command and when not quoted:

**if then else elif fi case esac for while until do done { }**

**Comments**

A word beginning with # and all the following characters up to a NEWLINE are ignored.

**Command Substitution**

The shell reads commands from the string between two grave accents (``) and the standard output from these commands may be used as all or part of a word. Trailing NEWLINE characters from the standard output are removed.

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" ... ` ... ` ... "), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a NEWLINE character (\NEWLINE), both the backslash and the NEWLINE are removed (see **Quoting**, later). In addition, backslashes used to escape dollar signs (\$) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, ‘, ", NEWLINE, and $ are left intact when the command string is read.

**Parameter Substitution**

The character $ is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

*name*=*value* [ *name*=*value* ] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter*}
: The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is ‘*’ or ‘@’, all the positional parameters, starting with **$1**, are substituted (separated by SPACE characters). Parameter **$0** is set from argument zero when the shell is invoked.

If the colon (:) is omitted from the following expressions, the shell only checks whether *parameter* is set or not.

${*parameter*:-*word*}
: If *parameter* is set and is nonnull, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
: If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
: If *parameter* is set and is nonnull, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message ‘**parameter null or not set**’ is printed.

${*parameter*:+*word*}
: If *parameter* is set and is nonnull, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

**echo** ${*d*:-‘**pwd**‘}

The following parameters are automatically set by the shell:

| | |
|---|---|
| # | The number of positional parameters in decimal. |
| - | Flags supplied to the shell on invocation or by the **set** command. |
| ? | The decimal value returned by the last synchronously executed command. |
| $ | The process number of this shell. |
| ! | The process number of the last background command invoked. |

The following parameters are used by the shell:

| | |
|---|---|
| **HOME** | The default argument (home directory) for the **cd** command. |
| **PATH** | The search path for commands (see **Execution** below). |
| **CDPATH** | The search path for the **cd** command. |
| **MAIL** | If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file. |

**MAILCHECK** This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each primary prompt.

**MAILPATH** A colon (:) separated list of filenames. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each filename can be followed by % and a message that will be printed when the modification time changes. The default message is '**you have mail**'.

**PS1** Primary prompt string, by default '$ '.

**PS2** Secondary prompt string, by default '> '.

**IFS** Internal field separators, normally SPACE, TAB, and NEWLINE.

**SHELL** When the shell is invoked, it scans the environment (see **Environment** below) for this name.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS**. **HOME** and **MAIL** are set by **login**(1).

**Blank Interpretation**

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

**Input/Output**

A command's input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple command* or may precede or follow a *command* and are *not* passed on to the invoked command. Note: parameter and command substitution occurs before *word* or *digit* is used.

*<word* Use file *word* as standard input (file descriptor 0).

*>word* Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

*>>word* Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the EOF); otherwise, the file is created.

*<<[-]word* After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an EOF. If, however, '-' is appended to:

- leading TAB characters are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*),
- leading TAB characters are stripped from the shell input as it is read and before each line is compared with *word*, and
- shell input is read up to the first line that literally matches the resulting *word*, or to an EOF.

If any character of *word* is quoted, (see **Quoting**, later), no additional processing is done to the shell input. If no characters of *word* are quoted:

- parameter and command substitution occurs,
- (escaped) \NEWLINE is ignored, and
- '\' must be used to quote the characters '\', '$', and '`'.

The resulting document becomes the standard input.

**<&***digit* Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using **>&***digit*.

**<&–** The standard input is closed. Similarly for the standard output using '**>&–**'.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (namely, file *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

Using the terminology introduced on the first page, under **Commands**, if a *command* is composed of several *simple commands*, redirection will be evaluated for the entire *command* before it is evaluated for each *simple command*. That is, the shell evaluates redirection for the entire *list*, then each *pipeline* within the *list*, then each *command* within each *pipeline*, then each *list* within each *command*.

If a command is followed by & the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

**Filename Generation**

Before a command is executed, each command *word* is scanned for the characters '*', '?', and '[ '. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The character '.' at the start of a filename or immediately following a '/', as well as the character '/' itself, must be matched explicitly.

- ***** Matches any string, including the null string.
- **?** Matches any single character.
- **[...]** Matches any one of the enclosed characters. A pair of characters separated by '–' matches any character lexically between the pair, inclusive. If the first character following the opening [ is a ! any character not enclosed is matched.

**Quoting**

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > NEWLINE SPACE TAB

A character may be *quoted* (made to stand for itself) by preceding it with a backslash (\) or inserting it between a pair of quote marks ('' or ""). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair \NEWLINE is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, "'").

Inside a pair of double quote marks (""), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If **$*** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces ("**$1 $2** ..."); however, if **$@** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces ("**$1" "$2"** ... ). \ quotes the characters \, ‘, ", and $. The pair \NEWLINE is removed before parameter and command substitution. If a backslash precedes characters other than \, ‘, ", $, and NEWLINE, then the backslash itself is quoted by the shell.

**Prompting**

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a RETURN is typed and further input is needed to complete a command, the secondary prompt (the value of **PS2**) is issued.

**Environment**

The *environment* (see **environ**(5V)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also '**set** –**a**'). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

**TERM=450** *cmd*

and

**(export TERM; TERM=450;** *cmd***)**

are equivalent (as far as the execution of *cmd* is concerned).

If the **–k** option is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set –k
echo a=b c
```

**Signals**

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent (but see also the **trap** command below). INTERRUPT is handled asynchronously.

**Execution**

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the **Special Commands** listed below, it is executed in the shell process. If the command name does not match a **Special Command**, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, .... are set to the arguments of the function. If the command name matches neither a **Special Command** nor the name of a defined function, a new process is created and an attempt is made to execute the command using **execve**(2V).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/usr/ucb:/bin:/usr/bin** (specifying **/usr/ucb**, **/bin**, and **/usr/bin**, in addition to the current directory). Directories are searched in order. The current directory is specified by a null path name, which can appear immediately after the equal sign (**PATH=:...**), between the colon delimiters (...**::**...) anywhere else in the path list, or at the end of the path list (...**:**). If the command name contains a / the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an binary executable (see **a.out**(5) for details) or an executable script (with a first line beginning with **#!**) it is assumed to be a file containing shell commands, and a subshell is spawned to read it. A parenthesized command is also executed in a subshell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary **execs** later). If the command was found in a relative directory, its location must be redetermined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the '**hash –r**' command is executed (see below).

**Special Commands**

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

**:** No effect; the command does nothing. A zero exit code is returned.

**.** *filename* Read and execute commands from *filename* and return. The search path specified by **PATH** is used to find the directory containing *filename*.

**break** [ *n* ] Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

**continue** [ *n* ] Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*'th enclosing loop.

**cd**[ *arg* ] Change the current directory to *argument*. The shell parameter **HOME** is the default *argument*. The shell parameter **CDPATH** defines the search path for the directory containing *argument*. Alternative directory names are separated by a colon (:). The default path is NULL (specifying the current directory). Note: the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *argument* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *argument*.

**echo** [ *argument* ... ]
Echo arguments. See **echo**(1V) for usage and description.

**eval** [ *argument* ... ]
The arguments are read as input to the shell and the resulting command(s) executed.

**exec** [ *argument* ... ]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, modify the shell's input/output.

**exit** [ *n* ] Exit a shell with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an EOF will also cause the shell to exit.)

**export** [ *name* ... ]
The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

**getopts** Use in shell scripts to parse positional parameters and check for legal options. See **getopts**(1) for usage and description.

**hash** [ **−r** ] [ *name* ... ]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **−r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *hits* is the number of times a command has been invoked by the shell process. *cost* is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *cost* will be incremented when the recalculation is done.

**login** [ *argument* ... ]
Equivalent to '**exec login** *argument*....' See **login**(1) for usage and description.

**newgrp** [ *argument* ... ]
Equivalent to '**exec newgrp** *argument*....' See **newgrp**(1) for usage and description.

**pwd** Print the current working directory. See **pwd**(1) for usage and description.

**read** [ *name* ... ]
One line is read from the standard input and, using the internal field separator, **IFS** (normally a SPACE or TAB character), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Lines can be continued using \NEWLINE. Characters other than NEWLINE can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is **0** unless an EOF is encountered.

**readonly** [ *name* ... ]
The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

**return** [ *n* ] Exit a function with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

**set** [ **–aefhkntuvx–** [ *argument* ... ] ]

**–a** Mark variables which are modified or created for export.

**–e** Exit immediately if a command exits with a nonzero exit status.

**–f** Disable filename generation.

**–h** Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).

**–k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.

**–n** Read commands but do not execute them.

**–t** Exit after reading and executing one command.

**–u** Treat unset variables as an error when substituting.

**–v** Print shell input lines as they are read.

**–x** Print commands and their arguments as they are executed.

**—** Do not change any of the options; useful in setting **$1** to '–'.

Using '+' rather than '–' turns off these options. These options can also be used upon invocation of the shell. The current set of options may be found in '$–'. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, and so on. If no arguments are given, the values of all names are printed.

**shift** [ *n* ] The positional parameters are shifted to the left, from position *n*+1 to position 1, and so on. Previous values for **$1** through **$***n* are discarded. If *n* is not given, it is assumed to be 1.

**test** Evaluate conditional expressions. See **test**(1V) for usage and description.

**times** Print the accumulated user and system times for processes run from the shell.

**trap** [ *arg* ] [ *n* ] ...

The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note: *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

**type** [ *name* ... ] For each *name*, indicate how it would be interpreted if used as a command name.

**umask** [ *ooo* ]

The user file-creation mode mask is set to *ooo* (see **csh**(1)). The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively. The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file. For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**). The current value of the mask is printed if *ooo* is omitted.

**unset** [ *name* ... ]
For each *name*, remove the corresponding variable or function. The variables **PATH, PS1, PS2, MAILCHECK** and **IFS** cannot be unset.

**wait** [ *n* ] Wait for the background process whose process ID is *n* and report its termination status. If *n* is omitted, all the shell's currently active background processes are waited for and the return code will be zero.

**EXIT STATUS**

Errors detected by the shell, such as syntax errors, return a nonzero exit status. If the shell is being used noninteractively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

**ENVIRONMENT**

The environment variables **LC_CTYPE, LANG,** and **LC_default** control the character classification throughout all command line parsing. These variables are checked in the following order: **LC_CTYPE, LANG,** and **LC_default.** When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE.** When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

**FILES**

**/etc/profile**
**$HOME/.profile**
**/tmp/sh***
**/dev/null**
**/usr/lib/rsh**

**SEE ALSO**

**cd**(1), **csh**(1), **echo**(1V), **env**(1), **getopts**(1), **login**(1), **newgrp**(1), **pwd**(1), **test**(1V), **wait**(1), **dup**(2V), **execve**(2V), **fork**(2V), **pipe**(2V), **sigvec**(2), **wait**(2V), **execl**(3V), **a.out**(5), **environ**(5V), **locale**(5)

*SunOS User's Guide: Doing More*

**WARNINGS**

Words used for filenames in input/output redirection are not interpreted for filename generation (see **File Name Generation**, above). For example, '**cat file1 > a***' will create a file named '**a***'.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message '**cannot fork, too many processes**', try using the **wait**(1) command to clean up your background processes. If this does not help, the system process table is probably full or you have too many active foreground processes. There is a limit to the number of process IDs associated with your login, and to the number the system can keep track of.

**BUGS**

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to **exec** the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For **wait** *n*, if *n* is not an active process ID, all the shell's currently active background processes are waited for and the return code will be zero.

## NAME

shelltool – run a shell (or other program) in a SunView terminal window

## SYNOPSIS

**shelltool** [ **–C** ] [ **–B** *boldstyle* ] [ **–I** *command* ] [ *generic-tool-arguments* ] [ *program* [ *arguments* ] ]

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**shelltool** is a standard SunView facility for shells or other programs that may use a standard tty-based interface.

When invoked, **shelltool** runs a program, (usually a shell) in an interactive terminal emulator based on a tty subwindow. Keyboard input is passed to that program. If the program is a shell, it accepts commands and runs programs in the usual way.

**cmdtool**(1), which provides moused-based editing, logging, and scrolling capabilities, also supports shell-level programs. See *SunView User's Guide* for more information.

To run graphics programs, use **gfxtool**(1).

## OPTIONS

**–C** Redirect system console output to this **shelltool**.

**–B** *boldstyle* Set the style for displaying bold text to *boldstyle*. *boldstyle* can be a string specifying one of the choices for the **/Tty/Bold_style** default, see **Defaults Options**, below, or it may be a numerical value for one of those choices, from 0 to 8, corresponding to the placement of the choice in the list.

**–I** *command* Pass *command* to the shell. SPACE characters within the command must be escaped.

*generic-tool-arguments*
**shelltool** accepts the generic tool arguments listed in **sunview**(1).

If a *program* argument is present, **shelltool** runs it. If no *program* is given, **shelltool** runs the program indicated by the **SHELL** environment variable, or **/usr/bin/sh** by default.

## USAGE

### Defaults Options

These options are available through **defaultsedit**(1).

**/Tty/Bold_style**

Select a style for emphasized text:

| | |
|---|---|
| **None** | Disable emphasis. |
| **Offset_X** | Thicken characters horizontally. |
| **Offset_Y** | Thicken characters vertically. |
| **Offset_X_and_Y** | Thicken characters both horizontally and vertically. |
| **Offset_XY** | Thicken characters diagonally. |
| **Offset_X_and_XY** | Thicken character both horizontally and diagonally. |
| **Offset_Y_and_XY** | Thicken characters both vertically and diagonally. |
| **Offset_X_and_Y_and_XY** | Thicken characters horizontally, vertically and diagonally. |
| **Invert** | Display emphasis as inverse video (the standard default). |

**/Tty/Inverse_mode**

Select a style for inverse video display:

| | |
|---|---|
| **Enable** | Enable inverse mode for inverted text. |
| **Disable** | Disable inverse mode for inverted text. |
| **Same_as_bold** | Display inverted text as bold text. |

**/Tty/Underline_mode**

Select a style for underlined text:

| | |
|---|---|
| **Enable** | Enable underline mode for underlined text. |
| **Disable** | Disable underline mode for underlined text. |
| **Same_as_bold** | Display underlined text as bold text. |

**/Tty/Retained**

When set to "Yes", hidden tty subwindow areas are retained in memory. This enhances the speed of repainting the screen at the expense of memory area. "No" is the standard default; it specifies that tty subwindows are not retained.

**The Terminal Emulator**

The tty subwindow is a terminal emulator. Whenever a tty subwindow is created, the startup file ~**/.ttyswrc** is read for initialization parameters that are specific to the tty subwindow.

**The .ttyswrc File**

A sample **.ttyswrc** file can be found in **/usr/lib/ttyswrc**. The command format for this file is:

| | |
|---|---|
| **#** | Comment. |
| **set** *variable* | Turn on the specified variable. |
| **mapi** *key text* | When *key* is typed pretend *text* was input. |
| **mapo** *key text* | When *key* is typed pretend *text* was output. |

The only currently defined variable is **pagemode**. *key* is one of L1-L15, F1-F15, T1-T15, R1-R15, LEFT, or RIGHT (see note below). *text* may contain escapes such as \E, \n, ^X, etc. (ESC, RETURN, and CTRL-X, respectively). See **termcap**(5) for the format of the string escapes that are recognized. Note: **mapi** and **mapo** may be replaced by another keymapping mechanism in the future.

When using the default kernel keyboard tables, the keys L1, LEFT, RIGHT, BREAK, R8, R10, R12, and R14 cannot be mapped in this way; they send special values to the tty subwindow. Also, when using the default kernel keyboard tables, L1-L10 are now used by SunView. See **input_from_defaults**(1) and **kbd**(4S) for more information on how to change the behavior of the keyboard.

It is possible to have terminal-based programs drive the tool in which its tty subwindow resides by sending special escape sequences. These escape sequences may also be sent by typing a key appropriately mapped using the **mapo** function described above. The following functions pertain to the tool in which the tty subwindow resides, not the tty subwindow itself.

| | |
|---|---|
| **\E[1t** | – open |
| **\E[2t** | – close (become iconic) |
| **\E[3t** | – move, with interactive feedback |
| **\E[3;TOP;LEFTt** | – move, to **TOP LEFT** (pixel coordinates) |
| **\E[4t** | – stretch, with interactive feedback |
| **\E[4;HT;WIDTHt** | – stretch, to **HT WIDTH** size (in pixels) |
| **\E[5t** | – front |
| **\E[6t** | – back |
| **\E[7t** | – refresh |
| **\E[8;ROWS;COLSt** | – stretch, to **ROWS COLS** size (in characters) |
| **\E[11t** | – report if open or iconic by sending **\E[1t** or **\E[2t** |
| **\E[13t** | – report position by sending **\E[3;TOP;LEFTt** |

| | |
|---|---|
| **\E[14t** | – report size in pixels by sending **\E[4;HT;WIDTHt** |
| **\E[18t** | – report size in characters by sending **\E[8;ROWS;COLSt** |
| **\E[20t** | – report icon label by sending **\E]Llabel\E** |
| **\E[21t** | – report tool header by sending **\E]llabel\E** |
| **\E]ltext\E** | – set tool header to text |
| **\E]lfile\E** | – set icon to the icon contained in file; file must be in *iconedit* output format |
| **\E]Llabel\E** | – set icon label to label |
| **\E[>OPT;...h** | – turn SB OPT on (**OPT** = 1 => pagemode), for example, **\E[>1;3;4h** |
| **\E[>OPT;...k** | – report **OPT**; sends **\E[>OPTl** or **\E[>OPTh** for each **OPT** |
| **\E[>OPT;...l** | – turn **OPT** off (**OPT** = 1 => pagemode), for example, **\E[>1;3;4l** |

See **EXAMPLES** for an example of using this facility.

**Selections**

Terminal subwindows support a selection facility that allows you to capture a block of text, move it between windows, and replicate it. You can make a selection by clicking the left button on the mouse at the top-left character of the block to capture, and then clicking the middle button on the bottom-right character. The selected text is highlighted. Multiple clicks of the LEFT mouse button capture:

| | |
|---|---|
| 1 click | a character |
| 2 clicks | a word |
| 3 clicks | a line |
| 4 clicks | a screenful |

You can also make a selection by moving the mouse while holding the select button, and then releasing it. The selection is deselected if you type any key or new output is written to the window that holds the selection.

**Menu**

To manipulate your selection, press the menu button over the terminal subwindow. A *ttysw* menu appears with the menu items discussed below:

**Copy, then Paste** When there is a selection in any window, the entire item is active. Selecting it copies the selection both to the clipboard and to the insertion point (cursor). It copies selections in tty, text, command, and panel subwindows, and It is intended to bridge the gap between **Stuff** and the selection facility (see *SunView User's Guide*. When there is no selection but there is text on the clipboard, only **Paste** is active. In this case, the contents of the clipboard are copied to the insertion point (cursor). When there is no selection and nothing on the clipboard, this item is inactive.

**Enable Page Mode**
**Disable Page Mode** Toggle page mode on and off. Page mode prevents output from scrolling off the screen. It is an alternative to **more**(1). When page mode is on, the cursor changes to resemble a tiny stop-sign when ever a screenful of output is displayed. To restart output, type any key, or select the **Continue** menu item that temporarily replaces **Enable Page Mode.**

**Stuff** is provided for backward compatibility. It copies the selection to the insertion point (cursor) as though they had been typed from the keyboard. **Stuff** can only handle selections made in a tty subwindow.

**Flush Input** Occasionally the input buffer fills up and the terminal emulator appears to freeze. If this happens, the '**Flush Input**' appears in the menu; choosing it clears the buffer and allows you to continue.

## EXAMPLES

The following aliases can be put into your ~/**.cshrc** file:

```
# dynamically set the name stripe of the tool:
alias header 'echo -n "\E]l\!*\E\"'
# dynamically set the label on the icon:
alias iheader 'echo -n "\E]L\!*\E\"'
# dynamically set the image on the icon:
alias icon 'echo -n "\E]I\!*\E\"'
```

## FILES

~/**.ttyswrc**
**/usr/lib/ttyswrc**
**/usr/demo/***

## SEE ALSO

**cmdtool**(1), **defaultsedit**(1), **gfxtool**(1), **input_from_defaults**(1), **more**(1), **rlogin**(1C), **sunview**(1), **kbd**(4S), **termcap**(5)

*SunView User's Guide*

## BUGS

If more than 256 characters are input to a terminal emulator subwindow without an intervening NEWLINE, the terminal emulator may hang. If this occurs, an alert will come up with a message saying '**Too many keystrokes in input buffer**'. Choosing the **Flush Input Buffer** menu item may correct the problem. This is a bug for a terminal emulator subwindow running on top of or **rlogin**(1C) to a machine with pre-4.0 release kernel.

**NAME**

size – display the size of an object file

**SYNOPSIS**

**size** [ *object-file* ... ]

**Sun386i SYNOPSIS**

**/usr/bin/size** [**–n**] [**–f**] [**–o**] [**–x**] [**–V**] *filename* ...

**DESCRIPTION**

**size** prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in hex and decimal, of each *object-file* argument. If no file is specified, **a.out** is used.

**Sun386i DESCRIPTION**

The Sun386i version of the System V compatibility package includes **/usr/bin/size**, which allows the System V options to be used, and creates the same output as the System V **size**(1) command; it produces section size information in bytes for each loaded section in COFF files. The size of the text, data, and bss (uninitialized data) sections is printed, as well as the sum of the sizes of these sections. If an archive file is given, it displays the information for all archive members.

**Sun386i OPTIONS**

**–n** Includes NOLOAD sections in the size.

**–f** Produces full output, that is, it prints the size of every loaded section, followed by the section name in parentheses.

**–o** Print numbers in octal, instead of the default which is decimal.

**–x** Print numbers in hexadecimal.

**–V** Supply version information.

**Sun386i DIAGNOSTICS**

**size:** *name*: **cannot open**
*name* cannot be read.

**size:** *name*: **bad magic**
*name* is not an appropriate common object file.

**Sun386i WARNINGS**

Since the size of bss sections is not known until link-edit time, this command does not give the true total size of pre-linked objects.

**SEE ALSO**

**cc**(1V), **size**(1), **a.out**(5), **ar**(5), **coff**(5)

**NAME**

sleep – suspend execution for a specified interval

**SYNOPSIS**

**sleep** *time*

**DESCRIPTION**

**sleep** suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

**SEE ALSO**

**sleep**(3V)

**BUGS**

*time* must be less than 2,147,483,647 seconds.

## NAME

snap – SunView application for system and network administration

## SYNOPSIS

**snap**

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

**snap** simplifies the execution of a variety of system administration tasks in the user-friendly environment of a SunView window. **snap** eases the following tasks: personal or centralized backup and restoration of files, management of users accounts and user groups, software installation, network administration, and management of devices such as printers, terminals, modems, and the peripheral box containing disk and tape drives.

Anyone can use **snap**, but the operations allowed depend on the secondary group membership of the user at the time that snap is invoked. There are four secondary user groups specifically recognized by **snap**, membership in which bestows various powers over the corresponding area of system administration. These are:

**accounts** users and user groups.

**devices** printers, terminals, modems, and peripheral box.

**operator** centralized backup and restoration of files, and installation of software.

**networks** domains and systems, including the New User Accounts feature and Automatic System Installation features.

A user's **snap** privileges depend upon which of these four groups he or she belongs to. If they get an account through New User Accounts, or if an administrator adds them using the defaults, new users become members of the primary group *users*, and are given all **snap** privileges. This can be changed by changing the secondary group membership of the primary group *users* with **snap**. Note: this does not change the group membership of existing users, but only of new users. The secondary group membership of existing users must be changed individually.

### Accounts

An administrator using **snap** can create new user accounts and remove existing ones, change a user's **snap** privileges, and control users' access to their accounts. New users can create their own accounts as they first login if the New User Accounts feature is activated as described under Networks below.

### Devices

Epson and Epson-like printers (most printers using the Centronics parallel interface), text serial printers, and HP Laserjet and compatible printers can be administered with **snap** . The supported terminal types are **vt-100** and **wyse.** The supported modem types are Hayes Smartmodem or a modem that is compatible with Hayes Smartmodem. For all other types of terminals, modems, or printers, the software must be configured manually. See *System and Network Administration* for details.

**snap** can add or remove, display and change information about, or disable or enable either a printer, a terminal, a modem, or the peripheral box containing disk and tape drives. Devices not added using **snap** can not be manipulated with **snap.**

**Operator**

Regardless of the primary or secondary group membership of users, they can backup and restore their own files with **snap**.

Backup and removal of all files can be done by members of the **operator** group.

**Networks**

Much of the network setup must be done when the first machine in the network, the master server, is started up, and when each client is connected and booted for the first time. Some of this information can never be changed.

Once the master and slave servers are installed, **snap** can be used to add and assign diskless clients to servers, remove them, modify their network roles, and perform all the functions listed above under Accounts, Devices, and Operator on any system in the network.

If desired, you can also enable or disable the feature that allows a user to create his own account while logging in (New User Accounts), and the automatic system installation feature, two possible security loopholes.

**SEE ALSO**

*Sun386i System and Network Administration*
*System and Network Administration*

## NAME

soelim – resolve and eliminate .so requests from nroff or troff input

## SYNOPSIS

**soelim** [ *filename* ... ]

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**soelim** reads the specified files or the standard input and performs the textual inclusion implied by the **nroff**(1) directives of the form

.so *somefile*

when they appear at the beginning of input lines. This is useful since programs such as **tbl**(1) do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of '–' is taken to be a file name corresponding to the standard input.

Note: inclusion can be suppressed by using ' ' ' instead of '.', that is,

**' so /usr/share/lib/tmac/tmac.s**

## EXAMPLE

A sample usage of **soelim** would be

**soelim exum?.n | tbl | nroff**

## SEE ALSO

**colcrt**(1), **more**(1), **nroff**(1), **tbl**(1)

## NAME

sort – sort and collate lines

## SYNOPSIS

**sort** [ **–bdfiMnr** ] [ **–t***c* ] [ *sort-field* ...] [ **–cmu** ] [ **–o**[ ]*output-file* ] [ **–T** *directory* ] [ **–y** *kmem* ] [ **–z** *recsz* ] *filename*...

## SYSTEM V SYNOPSIS

**/usr/5bin/sort** [ **–bdfiMnr** ] [ **–t***c* ] [ *sort-field* ...] [ **–cmu** ] [ **–o**[ ]*output-file* ] [ **–T** *directory* ] [ **–y** *kmem* ] [ **–z** *recsz* ] *filename*...

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

The **sort** program sorts and collates lines contained in the named files, and writes the result onto the standard output. If no *filename* argument is given, or if '–' appears as an argument, **sort** accepts input from the standard input.

Output lines are normally sorted on a character-by-character basis, from left to right within a line. The default collating sequence is the ASCII character set. Lines can also be sorted according to the contents of one or more fields specified by a *sort-field*, specification, using the *+sw* (starting-word), *–ew* (end-at-word), and the *–tc* (set-TAB-character/word delimiter) options, as described under OPTIONS below. When no word delimiter is specified, one or more adjacent white-space characters (SPACE and TAB) signify the end of the previous word; the lines:

```
^^^ xyz
^^^   xyz
```

are collated as:

```
^^^   xyz
^^^ xyz
```

Each *sort-field* is evaluated in command-line order; later fields are applied to the sorting sequence only when all earlier fields compare equally. When all specified fields compare equally between two or more lines, that subset of lines is sorted on a character-by-character basis, from left to right.

## SYSTEM V DESCRIPTION

When no fields are specified in the command line, the System V version of **sort** treats leading blanks as significant, even with the **–n** (numeric collating sequence) option; the lines:

```
123
 23
```

are collated as:

```
 23
123
```

## OPTIONS

### Collating Flags

**–b** Ignore leading SPACE characters when determining the starting and ending positions of a field.

**–d** Dictionary order. Only letters, digits and the white-space characters SPACE and TAB are significant in comparisons.

**–f** Fold in lower case. Treat upper- and lower-case letters equally in collating comparisons.

**–i** Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

**–M** Month order. The first three non-blank characters of the field are folded to upper case and collated according to the sequence: **JAN FEB ... DEC**. Field values outside this range appear earlier than **JAN**. The **–M** option implies the **–b** option.

**–n** Numeric collating sequence. An initial numeric string, consisting of optional blanks, optional minus signs, and zero or more digits with an optional decimal point, is sorted by arithmetic value. The **–n** option implies the **–b** option, but only when at least one *sort-field* is specified on the command line.

**–r** Reverse the current collating sequence.

**Field Specification Options**

**–t***c* Use *c* as the word delimiter character; unlike white-space characters, adjacent delimiters indicate word breaks; if **:** is the delimiter character, **::** delimits an empty word.

*sort-field*

This is a combination of options that specifies a field, within each line, to sort on. A *sort-field* specification can take either of the following forms:

*+sw*[*cf*]
*+sw –ew*[*cf*]

where *sw* is the number of the starting word (beginning with '0') to include in the field, *ew* is the number of the word before which to end the field, and *cf* is a string containing collating flags (without a leading '–'.) When included in a *sort-field* specification, these flags apply only to the field being specified, and when given, override other collating flags given in separate arguments (which otherwise apply to an entire line).

If the *–ew* option is omitted, the field continues to the end of a line.

You can apply a character offset to *sw* and *ew* to indicate that a field is to start or end a given number of characters within a word, using the notation: '*w.c*'. A starting position specified in the form: '*+w.c*' indicates the character in position *c* (beginning with **0** for the first character), within word *w* (**1** and **1.0** are equivalent). An ending position specified in the form: '*–w.c*' indicates that the field ends at the character just prior to position *c* (beginning with **0** for the delimiter just prior to the first character), within word *w*. If the **–b** flag is in effect, *c* is counted from the first non-white-space or non-delimiter character in the field, otherwise, delimiter characters are counted.

**Other Options**

**–c** Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

**–m** Merge only, the input files are already sorted.

**–u** Unique. Emit only the first line in each set of lines for which all sorting fields compare equally.

**–o***output-file*
**–o** *output-file*

Direct output to the file specified as *output-file*, instead of the standard output. This file may be the same as one of the input files.

**–y** *kmem*

The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, **sort** begins using a system default memory size, and continues to add space as needed. If this option is given **sort** starts with *kmem*, kilobytes of memory, if allowed, or as close to that amount as possible. Supplying **–y0** guarantees that **sort** starts with a minimum of memory. By convention, **–y** (with no argument) starts with maximum memory.

**–z** *recsz*

The size of the longest line read is recorded in the sort phase so that buffers can be allocated during the merge phase. If the sort phase is omitted because either of the **–c** or **–m** options is in effect, a default size of 1024 bytes is used. Lines longer than the buffer size terminate **sort** abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) avoids this.

**–T** *directory*

The *directory* argument is the name of a directory in which to place temporary files.

## EXAMPLES

Sort the contents of **input-file** with word number 1 (the second word) as the sort key:

**sort +1 –2 input-file**

Sort, in reverse order, the contents of **input-file1** and **input-file2**, placing the output in **output-file** and using the first character of the second field as the sort key:

**sort –r –o output-file +1.0 –1.1 input-file1 input-file2**

Sort, in reverse order, the contents of **input-file1** and **input-file2** using the first non-blank character of the second field as the sort key:

**sort –r +1.0b –1.1b input-file1 input-file2**

Print the password file (**passwd**(5)) sorted by the numeric user ID (the third colon-separated field):

**sort –t: +2n –3 /etc/passwd**

Print the lines of the already sorted file **input-file**, suppressing all but the first occurrence of lines having the same third field (the options **–mu** with just one input file make the choice of a unique representative from a set of equal lines predictable):

**sort –mu +2 –3 input-file**

## FILES

**/usr/tmp/stm???**

## SEE ALSO

**comm**(1), **join**(1), **rev**(1), **uniq**(1)

## DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (such as when input lines are too long), and for disorders discovered under the **–c** option.

When the last line of an input file is missing a NEWLINE, **sort** appends one, prints a warning message, and continues.

## NAME

sortbib – sort a bibliographic database

## SYNOPSIS

**sortbib** [ –s*KEYS* ] *database*...

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**sortbib** sorts files of records containing **refer** key-letters by user-specified keys. Records may be separated by blank lines, or by '**.[**' and '**.]**' delimiters, but the two styles may not be mixed together. This program reads through each *database* and pulls out key fields, which are sorted separately. The sorted key fields contain the file pointer, byte offset, and length of corresponding records. These records are delivered using disk seeks and reads, so **sortbib** may not be used in a pipeline to read standard input.

By default, **sortbib** alphabetizes by the first **%A** and the **%D** fields, which contain the senior author and date. The –s option is used to specify new *KEYS*. For instance, –**sATD** will sort by author, title, and date, while –**sA+D** will sort by all authors, and date. Sort keys past the fourth are not meaningful. No more than 16 databases may be sorted together at one time. Records longer than 4096 characters will be truncated.

**sortbib** sorts on the last word on the **%A** line, which is assumed to be the author's last name. A word in the final position, such as '**jr.**' or '**ed.**', will be ignored if the name beforehand ends with a comma. Authors with two-word last names or unusual constructions can be sorted correctly by using the **nroff** convention '**\0**' in place of a blank. A **%Q** field is considered to be the same as **%A**, except sorting begins with the first, not the last, word. **sortbib** sorts on the last word of the **%D** line, usually the year. It also ignores leading articles (like '**A**' or '**The**') when sorting by titles in the **%T** or **%J** fields; it will ignore articles of any modern European language. If a sort-significant field is absent from a record, **sortbib** places that record before other records containing that field.

## SEE ALSO

**addbib**(1), **indxbib**(1), **lookbib**(1), **refer**(1), **roffbib**(1)

**refer** in *Formatting Documents*

## BUGS

Records with missing author fields should probably be sorted by title.

## NAME

spell, hashmake, spellin, hashcheck – report spelling errors

## SYNOPSIS

**spell** [ **–blvx** ] [ **–d** *hlist* ] [ **–h** *spellhist* ] [ **–s** *hstop* ] [ *+local_file* ] [ *filename* ] . . .

**/usr/lib/spell/hashmake**

**/usr/lib/spell/spellin** *n*

**/usr/lib/spell/hashcheck** *spelling_list*

## DESCRIPTION

**spell** collects words from the named files, and looks them up in a hashed spelling list. Words that do not appear in the list, or cannot be derived from those that do appear by applying certain inflections, prefixes or suffixes, are displayed on the standard output.

If there are no *filename* arguments, words to check are collected from the standard input. **spell** ignores most **troff**(1), **tbl**(1), and **eqn**(1) constructs. Copies of all output words are accumulated in the history file, and a *stop* list filters out misspellings (for example, their=thy–y+ier) that would otherwise pass.

By default, **spell** (like **deroff**(1)) follows chains of included files (**.so** and **.nx troff**(1) requests), *unless* the names of such included files begin with **/usr/lib**.

If a *+local_file* argument is specified, words found in *local_file* are removed from **spell**'s output. *local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to **spell**'s own spelling list) for each job.

The standard spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

Three programs help maintain and check the hash lists used by *spell*:

**hashmake** Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

**spellin** Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.

**hashcheck** Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

## OPTIONS

**–b** Check British spelling. Besides preferring "centre", "colour", "programme", "speciality", "travelled", and so on, this option insists upon *-ise* in words like *standardize*, despite what Fowler and the OED say.

**–l** Follow the chains of *all* included files.

**–v** Print all words not literally in the spelling list, as well as plausible derivations from spelling list words.

**–x** Print every plausible stem with '=' for each word.

**–d** *hlist* Use the file *hlist* as the hashed spelling list.

**–h** *spellhist* Place misspelled words with a user/date stamp in file *spellhist*.

**–s** *hstop* Use *hstop* as the hashed stop list.

**FILES**

| | |
|---|---|
| **/usr/lib/spell/hlist[ab]** | hashed spelling lists, American & British |
| **/usr/lib/spell/hstop** | hashed stop list |
| **/usr/lib/spell/spellhist** | history file |
| **/usr/lib/spell/spellprog** | program called by the **/usr/bin/spell** shell script |

**SEE ALSO**

**deroff**(1), **sed**(1V), **sort**(1V), **tee**(1)

**BUGS**

The spelling list's coverage is uneven; new installations may wish to monitor the output for several months to gather local additions.

British spelling was done by an American.

**NOTES**

Misspelled words can be monitored by default by setting the **H_SPELL** variable in **/usr/bin/spell** to the name of a file that has permission mode 666.

**spell** works only on English words defined in the US ASCII codeset.

## NAME

spline – interpolate smooth curve

## SYNOPSIS

**spline** [ **–aknpx** ] ...

## DESCRIPTION

**spline** takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by **graph**(1G).

## OPTIONS

**–a** Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be **1** if next argument is not a number.

**–k** The constant $k$ used in the boundary value computation

$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}'' \quad (101)$$

is set by the next argument. By default $k = 0$.

**–n** Space output points so that approximately $n$ intervals occur between the lower and upper $x$ limits. (Default $n = 100$.)

**–p** Make output periodic, that is, match derivatives at ends. First and last input values should normally agree.

**–x** Next 1 (or 2) arguments are lower (and upper) $x$ limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

## SEE ALSO

**graph**(1G)

R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed.

## DIAGNOSTICS

When data is not strictly monotonic in $x$, **spline** reproduces the input without interpolating extra points.

## BUGS

A limit of 1000 input points is enforced silently.

**NAME**

split – split a file into pieces

**SYNOPSIS**

**split** [ *–number* ] [ *infile* [ *outfile* ] ]

**DESCRIPTION**

**split** reads *infile* and writes it in *number*-line pieces (default 1000) onto a set of output files (as many files as necessary). The name of the first output file is *outfile* with **aa** appended, the second file is *outfile* **ab**, and so on lexicographically.

If no *outfile* is given, **x** is used as default (output files will be called **xaa**, **xab**, etc.).

If no *infile* is given, or if '–' is given in its stead, then the standard input file is used.

**OPTIONS**

*–number*

Number of lines in each piece.

**NAME**

strings – find printable strings in an object file or binary

**SYNOPSIS**

**strings** [ – ] [ –**o** ] [ –*number* ] *filename* . . .

**DESCRIPTION**

**strings** looks for ASCII strings in a binary file. A string is any sequence of 4 or more printing characters ending with a NEWLINE or a null character.

**strings** is useful for identifying random object files and many other things.

**OPTIONS**

– Look everywhere in the file for strings. If this flag is omitted, **strings** only looks in the initialized data space of object files.

–**o** Precede each string by its offset in the file.

–*number*

Use *number* as the minimum string length rather than 4.

**SEE ALSO**

od(1V)

**NOTES**

**strings** is not 8-bit clean because it makes too many mistakes when it is expected to look for strings containing non-ASCII characters.

**BUGS**

The algorithm for identifying strings is extremely primitive.

**NAME**

strip – remove symbols and relocation bits from an object file

**SYNOPSIS**

**strip** *filename...*

**DESCRIPTION**

**strip** removes the symbol table and relocation bits ordinarily attached to the output of the assembler and linker. This is useful to save space after a program has been debugged.

The effect of **strip** is the same as use of the **–s** option of **ld**(1).

**SEE ALSO**

**ld**(1), **a.out**(5)

**BUGS**

Unstripped 2.0 binary files will not run if stripped by the 3.0 version. A message of the form:

**pid** *xxx*: **killed due to swap problems in I/O error mapping page.**

when attempting to run a program indicates that this is the problem.

## NAME

stty – set or alter the options for a terminal

## SYNOPSIS

**stty** [ **–ag** ] [ *option* ] ...

## SYSTEM V SYNOPSIS

**/usr/5bin/stty** [ **–ag** ] [ *option* ] ...

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**stty** sets certain terminal I/O options for the device that is the current standard output. Without arguments, it reports the settings of certain terminal options for the device that is the standard output; the settings are reported on the standard error.

Detailed information about the modes listed in the first five groups below may be found in **termio**(4). Options in the last group are implemented using options in the previous groups. Note: many combinations of options make no sense, but no sanity checking is performed.

## SYSTEM V DESCRIPTION

**stty** sets or reports terminal options for the device that is the current standard input; the settings are reported on the standard output.

## OPTIONS

**–a** Report all of the option settings.

**–g** Report current settings in a form that can be used as an argument to another **stty** command.

### Special Requests

**speed** The terminal speed alone is printed on the standard output.

**size** The terminal (window) sizes are printed on the standard output, first rows and then columns.

**size** and **speed** always report on the settings of **/dev/tty**, and always report the settings to the standard output.

### Control Modes

[–]**parenb** Enable parity generation and detection. With a '–', disable parity checking.

[–]**parodd** Select odd parity. With a '–', select even parity.

**cs5 cs6 cs7 cs8**
Select character size.

**0** Hang up phone line immediately.

**50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 19200 exta 38400 extb**
Set terminal baud rate to the number given, if possible. (Not all speeds are supported by all hardware interfaces.)

[–]**hupcl** Hang up connection on last close. With a '–', do not hang up connection.

[–]**hup** Same as **hupcl**.

[–]**cstopb** Use two stop bits per character. With a '–', use one stop bit per character.

[–]**cread** Enable the receiver. With a '–', disable the receiver.

[–]**clocal** Assume a line without modem control. With a '–', assume a line with modem control.

**Input Modes**

**[–]ignbrk** Ignore break on input. With a '–', do not ignore a break on input.

**[–]brkint** Signal SIGINT on break. With a '–', do not signal.

**[–]ignpar** Ignore parity errors. With a '–', do not ignore parity errors.

**[–]parmrk** Mark parity errors With a '–', do not mark parity errors.

**[–]inpck** Enable input parity checking. With a '–', disable input parity checking.

**[–]istrip** Strip input characters to seven bits. With a '–', do not strip input characters.

**[–]inlcr** Map NEWLINE to RETURN on input. With a '–', do not map on input.

**[–]igncr** Ignore RETURN on input. With a '–', do not ignore RETURN on input.

**[–]icrnl** Map RETURN to NEWLINE on input. With a '–', do not map.

**[–]iuclc** Map upper-case alphabetics to lower case on input. With a '–', do not map.

**[–]ixon** Enable START/STOP output control. With a '–', disable output control. When enabled, output is stopped by sending a STOP character and started by sending a START character.

**[–]ixany** Allow any character to restart output. With a '–', only restart with a START character.

**[–]decctlq** Same as **–ixany**.

**[–]ixoff** Request that the system send START/STOP characters when the input queue is nearly empty/full. With a '–', request that the system not send START/STOP characters.

**[–]tandem** Same as **ixoff**.

**[–]imaxbel** Request that the system send a BEL character to your terminal, and not to flush the input queue, if a character received when the input queue is full. With a '–', request that it flush the input queue and not send a BEL character.

**[–]iexten** Enable all SunOS special characters, such as word erase. With a '–', enable only the POSIX subset of special characters (**INTR, QUIT, ERASE, KILL, EOF, NL, EOL, SUSP, STOP, START,** and **CR**).

**Output Modes**

**[–]opost** Post-process output. With a '–', do not post-process output; ignore all other output modes.

**[–]olcuc** Map lower-case alphabetics to upper case on output. With a '–', do not map.

**[–]onlcr** Map NEWLINE to RETURN-NEWLINE on output. With a '–', do not map.

**[–]ocrnl** Map RETURN to NEWLINE on output. With a '–', do not map.

**[–]onocr** Do not place RETURN characters at column zero. With a '–', do place RETURN characters at column zero.

**[–]onlret** On the terminal NEWLINE performs the RETURN function. With a '–', NEWLINE does not perform the RETURN function.

**[–]ofill** Use fill characters for delays. With a '–', use timing for delays.

**[–]ofdel** Fill characters are DEL characters. With a '–', fill characters are NUL characters.

**cr0 cr1 cr2 cr3**
Select style of delay for RETURN characters.

**nl0 nl1** Select style of delay for LINEFEED characters.

**tab0 tab1 tab2 tab3**
Select style of delay for horizontal TAB characters.

**bs0 bs1** Select style of delay for BACKSPACE characters.

**ff0 ff1** Select style of delay for form FORMFEED characters.

**vt0 vt1** Select style of delay for vertical TAB characters.

**Local Modes**

**[–]isig** Enable the checking of characters against the special characters **INTR** and **QUIT**. With a '–', disable this checking.

**[–]icanon** Enable canonical input (**ERASE**, **KILL**, **WERASE**, and **RPRNT** processing). With a '–', disable canonical input.

**[–]cbreak** Same as **–icanon**.

**[–]xcase** Perform canonical upper/lower-case presentation. With a '–', do not perform canonical upper/lower-case presentation.

**[–]echo** Echo back every character typed. With a '–', do not echo back.

**[–]echoe** Echo the **ERASE** character as a sequence of BACKSPACE-SPACE-BACKSPACE. With a '–', echo the **ERASE** character as itself.

**[–]crterase** Same as **echoe**.

**[–]echok** Echo NEWLINE after echoing a **KILL** character. With a '–', do not echo NEWLINE after echoing a **KILL** character.

**lfkc** Same as **echok**; obsolete.

**[–]echonl** Echo NEWLINE, even if **echo** is not set. With a '–', do not echo NEWLINE if **echo** is not set.

**[–]noflsh** Disable flush after **INTR** or **QUIT**. With a '–', enable flush.

**[–]tostop** Stop background jobs that attempt to write to the terminal. With a '–', allow background jobs to write to the terminal.

**[–]echoctl** Echo control characters as **x** (and delete as '**?**'.) Print two BACKSPACE characters following the **EOF** character (default CTRL-D). With a '–', echo control characters as themselves.

**[–]ctlecho** Same as **echoctl**.

**[–]echoprt** Echo erased characters backwards within '\' and '/'; used on printing terminals. With a '–', echo erased characters as indicated by **echoe**.

**[–]prterase** Same as **echoprt**.

**[–]echoke** Echo the **KILL** character by erasing each character on the line as indicated by **echoprt** and **echoe**. With a '–', echo the **KILL** character as indicated by **echoctl** and **echok**.

**[–]crtkill** Same as **echoke**.

*control-character c*
Set *control-character* to *c*, where *control-character* is one of **erase**, **kill**, **intr**, **quit**, **eof**, **eol**, **eol2**, **start**, **stop**, **susp**, **rprnt**, **flush**, **werase**, or **lnext**. If *c* is preceded by a caret (^), (escaped from the shell) then the value used is the corresponding CTRL character (for instance, "**^D**' is a CTRL-D); "**^?**' is interpreted as DEL and "**^–**' is interpreted as undefined.

**min** *i* Set the MIN value to *i*.

**time** *i* Set the TIME value to *i*.

**rows** *n* Set the recorded number of rows on the terminal to *i*.

**columns** *i* Set the recorded number of columns on the terminal to *i*.

**cols** *i* An alias for **columns** *i*.

**Combination Modes**

**cooked** Process the **ERASE**, **WERASE**, **KILL**, **INTR**, **QUIT**, **EOF**, **EOL**, **EOL2**, **STOP**, **START**, **SUSP**, **RPRNT**, **FLUSH**, and **LNEXT** characters specially, and perform output post-processing.

**evenp** or **parity**
Enable **parenb**, disable **parodd**, and set **cs7**.

**oddp** Enable **parenb** and **parodd**, and set **cs7**.

**–evenp** or **–parity**
Disable **parenb**, and set **cs8**.

**–oddp** Disable **parenb** and **parodd**, and set **cs8**.

**pass8** Disable **parenb** and **istrip**, and set **cs8**.

**–pass8** Enable **parenb** and **istrip**, and set **cs7**.

**litout** Disable **parenb**, **istrip**, and **opost**, and set **cs8**.

**–litout** Enable **parenb**, **istrip**, and **opost**, and set **cs7**.

[–]**raw** Enable raw input and output. With a '–', disable raw I/O. In raw mode, there is no special processing of the **ERASE**, **WERASE**, **KILL**, **INTR**, **QUIT**, **EOF**, **EOL**, **EOL2**, **STOP**, **START**, **SUSP**, **RPRNT**, **FLUSH**, nor **LNEXT** characters, nor is there any other input pre-processing nor output post-processing. **brkint**, **istrip**, **imaxbel**, and **parenb** are disabled, and **cs8** is set.

[–]**nl** Unset **icrnl**, **onlcr**. With a '–', set them. In addition **–nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

[–]**lcase** Set **xcase**, **iuclc**, and **olcuc**. With a '–', unset them.

[–]**LCASE** Same as **lcase** (**–lcase**).

[–]**tabs**
**tab3** Preserve TAB characters when printing. With a '–', or with **tab3**, expand TAB characters to SPACE characters.

**ek** Reset the **ERASE** and **KILL** characters back to normal: **DEL** and CTRL-**U**).

**sane** Reset all modes to some reasonable values.

**crt** Set options for a CRT (**echoe**, **echoctl**, and, if >= 1200 baud, **echoke**.)

**dec** Set all modes suitable for Digital Equipment Corp. operating systems users (**ERASE**, **KILL**, and **INTR** characters to ^?, ^U, and ^C, **decctlq**, and **crt**.)

**term** Set all modes suitable for the terminal type **term**, where **term** is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**. **-crtscts** Raise the RTS (Request to Send) modem control line. Suspends output until the CTS (Clear to Send) line is raised.

**ENVIRONMENT**

The environment variables LC_CTYPE, LANG, and LC_**default** control the character classification throughout **stty**. On entry to **stty**, these environment variables are checked in the following order: LC_CTYPE, LANG, and LC_**default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for LANG does not override the current valid character classification rules of LC_CTYPE. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

**SEE ALSO**

**ioctl**(2), **termio**(4), **locale**(5)

**NAME**

stty_from_defaults – set terminal editing characters from the defaults database

**SYNOPSIS**

**stty_from_defaults**

**AVAILABILITY**

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**stty_from_defaults** is a utility provided with the *SunView* environment.

**stty_from_defaults** sets the three editing characters (to erase a character, erase a word, and kill a line) according to the choices in your defaults database. It does not set any other tty options. If you run **stty**(1V) in your **.login** or **rc.local** files, you may want to run **stty_from_defaults** immediately after it. This will override any settings of *stty*erase, *stty*werase, or *stty*kill, so that you will have the same character-editing behavior with SunView application programs.

To specify the editing characters **stty_from_defaults** will set, run **defaultsedit**(1) and select the "Text" category. The editing characters are called Edit_back_char, Edit_back_word, and Edit_back_line. Type the value you want to the right of each item. To specify CTRL-X, type '**X**'— that is, the three characters '\', '^', and '**X**'. To specify DEL, type '**?**'.

If you do not specify your own values, the default values are DEL, CTRL-**W**, and CTRL-**U**, respectively.

**SEE ALSO**

**defaultsedit**(1), **stty**(1V), **sunview**(1)

*SunView User's Guide*

## NAME

su – super-user, temporarily switch to a new user ID

## SYNOPSIS

**su** [ – ] [ –**f** ] [ *username* [ *arg*... ] ]

## SYSTEM V SYNOPSIS

**su** [ – ] [ *username* [ *arg*... ] ]

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**su** creates a new shell process that has the user ID for the specified *username* as its real and effective user ID. **su** asks for the password, just as if you were logging in as *username*, and, if the password is given, changes the real and effective user IDs and group IDs and group set to those of *username* and invokes the shell specified in the password file for that *username*, without changing the current directory. The user environment is thus unchanged except for **HOME** and **SHELL**, which are taken from the password file for the user being substituted (see **environ**(5V)). If *username* is not **root**, **USER** is changed to *username*. The new user ID stays in force until the shell exits.

The new shell will not be a login shell, so it will not read *username*'s **.login** or **.profile** files, but it will read any other configuration files for that user (for instance, the **.cshrc** file for the C shell) just as if that user had invoked a new shell.

If no *username* is specified, **root** is assumed. If the **wheel** group (group 0) does not contain a null user list and has members, only they can **su** to **root**, even with the root password. To remind the super-user of his responsibilities, the shell substitutes '#' for '$' or '%' in its usual prompt (except if you will be running **sh**(1) and **PS1** is set - in this case, the prompt will not be modified). If *arg*s are given, they are passed to *username*'s shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like **sh**(1) and **csh**(1), an *arg* of the form –**c** *string* executes *string* via the shell.

## OPTIONS

| | |
|---|---|
| – | Perform a complete login. Remove all variables from the environment except for **TERM**, set **USER** to *username*, set **HOME** and **SHELL** as specified above, set **PATH** to **:/usr/ucb:/bin:/usr/bin**, change directories to *username*'s home directory, and tell the shell to read *username*'s **.login** or **.profile** file. |
| –**f** | Perform a fast **su** by passing the –**f** flag to the shell. This flag is only meant for use with the C shell; it will prevent the C shell from reading *username*'s **.cshrc** file. If it is used with the Bourne shell, it will disable filename generation. |

## FILES

**.cshrc**
**.login**
**.profile**

## SEE ALSO

**csh**(1), **login**(1), **sh**(1), **environ**(5V)

## NOTES

**su** does not accept 8-bit user IDs. See **login**(1) for explanations about why 8-bit **login** names are not acceptable.

## BUGS

**su** fails when run from within a subdirectory of a directory that *username* either cannot search, or cannot read (that is, *username* does not have both read and execute permission).

**su** fails to reset the user ID to root when the current working directory is in an NFS-mounted file system, and does not have its search permission set for "other" users.

## NAME

sum – calculate a checksum for a file

## SYNOPSIS

**sum** *filename*

## SYSTEM V SYNOPSIS

**/usr/5bin/sum** [ **–r** ] *filename*

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**sum** calculates and displays a 16-bit checksum for the named file, and also displays the size of the file in kilobytes. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The checksum is calculated by an algorithm which may yield different results on machines with 16-bit **int**s and machines with 32-bit **int**s, so it cannot always be used to validate that a file has been transferred between machines with different-sized **int**s.

## SYSTEM V DESCRIPTION

**sum** calculates and prints a 16-bit checksum for the named file, and also prints the number of 512-byte blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. This algorithm is independent of the size of **int**s on the machine.

## SYSTEM V OPTIONS

The option **–r** causes the (machine-dependent) algorithm used by the non-System V **sum** to be used in computing the checksum.

## SEE ALSO

**wc**(1)

## DIAGNOSTICS

**Read error** is indistinguishable from EOF on most devices; check the block count.

## NAME

sunview – the SunView window environment

## SYNOPSIS

**sunview** [ **–i** ] [ **–p** ] [ **–B** | **–F** | **–P** ] [ **–S** ] [ **–8bit_color_only** ] [ **–overlay_only** ] [ **–toggle_enable** ]
[ **–b** *red green blue* ] [ **–d** *display-device* ] [ **–f** *red green blue* ] [ **–k** *keyboard-device* ]
[ **–m** *mouse-device* ] [ **–n** | **–s** *startup-filename* ] [ **–background** *raster-filename* ]
[ **–pattern on** | **off** | **gray** | *iconedit-filename* ]

## DESCRIPTION

**sunview** starts up the SunView environment and (unless you have specified otherwise) a default layout of a few useful "tools," or window-based applications.

See **Start-up Processing** below to learn how to specify your own initial layout of tools. Some of the behavior of **sunview** is controlled by settings in your defaults database; see **SunView Defaults** below, and **defaultsedit**(1) for more information.

To exit **sunview** use the **Exit SunView** menu item. In an emergency, type CTRL-D then CTRL-Q (there is no confirmation in this case).

## OPTIONS

**–i** Invert the background and foreground colors used on the screen. On a monochrome monitor, this option provides a video reversed image. On a color monitor, colors that are not used as the background and foreground are not affected.

**–p** Print to the standard output the name of the window device used for the **sunview** background.

**–B** Use the "background color" ( **–b** ) for the background.

**–F** Use the "foreground color" ( **–f** ) for the background.

**–P** Use a stipple pattern for the background. This option is assumed unless **–F** or **–B** is specified.

**–S** Set **Click-to-type** mode, allowing you to select a window by clicking in it. Having done so, input is directed to that window regardless of the position of the pointer, until you click to select some other window.

**–8bit_color_only**
For multiple plane group frame buffers, only let windows be created in the 8 bit color plane group. This frees up the black and white overlay plane to have a separate desktop running on it. This option is usually used with the **–toggle_enable** option. See **Multiple Desktops on the Same Screen**, below.

**–overlay_only** For multiple plane group frame buffers, only let windows be created in the black and white overlay plane group. This frees up the 8 bit color plane group to have a separate desktop running in it. This option is usually used with the **–toggle_enable** option. See **Multiple Desktops on the Same Screen**, below.

**–toggle_enable** For multiple plane group frame buffers, when sliding the pointer between different desktops running within different plane groups on the same screen, change the enable plane to allow viewing of the destination desktop. See **Multiple Desktops on the Same Screen**, below.

**–b** *red green blue*

Specify values for the *red*, *green* and *blue* components of the background color. If this option is not specified, each component of the background color is 255 (white). Sun 3/110 system users that use this option should use the **–8bit_color_only** option as well.

**–d** *display-device* Use *display-device* as the output device, rather than **/dev/fb** the default frame buffer device.

**–f** *red green blue* Specify values for the *red*, *green* and *blue* components of the foreground color. If this option is not specified, each component of the foreground color is 0 (black). Sun 3/110 system users that use this option should use the **–8bit_color_only** option as well.

**–k** *keyboard-device*

Accept keyboard input from *keyboard-device*, rather than **/dev/kbd**, the default keyboard device.

**–m** *mouse-device* Use *mouse-device* as the system pointing device (locator), rather than **/dev/mouse**, the default mouse device.

**–n** Bypass startup processing by ignoring the **/usr/lib/.sunview** and **~/.sunview** (and **~/.suntools**) files.

**–s** *startup-filename*

Read startup commands from *startup-filename* instead of **/usr/lib/.sunview** or **~/.sunview**).

**–background** *raster-filename*

Use the indicated raster file as the image in your background. The raster file can be created with **screendump**(1). Screen dumps produced on color monitors currently do not work as input to this option. Small images are centered on the screen.

**–pattern on | off | gray |** *iconedit-filename*

Use the indicated "pattern" to cover the background. **on** means to use the default desktop gray pattern. **off** means to not use the default desktop gray pattern. **gray** means to use a 50% gray color on color monitors. *iconedit-filename* is the name of a file produced with **iconedit**(1) which contains an image that is to be replicated over the background.

## USAGE

### Windows

The SunView environment always has one window open, referred to as the background, which covers the whole screen. A solid color or pattern is its only content. Each application is given its own window which lies on top of some of the background (and possibly on top of other applications). A window obscures any part of another window which lies below it.

### Input to Windows

Mouse input is always directed to the window that the pointer is in at the time. Keyboard input can follow mouse input or, it can remain within a designated window using the **Click-to-Type** default setting. If you are not using **Click-to-Type**, and the pointer is on the background, keyboard input is discarded. Input actions (mouse motions, button clicks, and keystrokes) are synchronized, which means that you can "type-ahead" and "mouse-ahead," even across windows.

### Mouse Buttons

LEFT mouse button Click to select or choose objects.

MIDDLE mouse button In text, click once to shorten or lengthen your selection. In graphic applications or on the desktop, press and hold to move objects.

RIGHT mouse button Press and hold down to invoke menus.

**Menus**

**sunview** provides pop-up menus. There are two styles of pop-up menus: an early style, called "stacking menus," and a newer style, called "walking menus" (also known as "pull-right menus"). In the current release, walking menus are the default; stacking menus are still available as a defaults option.

Usually, a menu is invoked by pressing and holding the RIGHT mouse button. The menu remains on the screen as long as you hold the RIGHT mouse button down. To choose a menu item, move the pointer onto it (it is then highlighted), then release the RIGHT mouse button.

Another available option is "stay-up menus." A stay-up menu is invoked by pressing and releasing the RIGHT mouse button. The menu appears on the screen after you release the RIGHT mouse button. To choose a menu item, move the pointer onto it (it is then highlighted), then press and release the RIGHT mouse button a second time. Stay-up menus are an option in your defaults database; see **SunView Defaults** below.

With walking menus, any menu item can have an arrow pointing (⇒) to the right. Moving the pointer onto this arrow pops up a "sub-menu," with additional items. Choosing the item with an arrow (the "pull-right item") invokes the first item on the sub-menu.

**The SunView Menu**

You can use the default SunView menu to start SunView applications and perform some useful functions. To invoke it, hold down the RIGHT mouse button when the pointer is anywhere in the background.

The default SunView menu is defined in the file **/usr/lib/.rootmenu**. It consists of four sub-menus, labeled **Shells**, **Editors**, **Tools**, and **Services**, along with an **Exit** SunView item. These sub-menus contain the following items:

**Shells**

**Command Tool** Bring up a **cmdtool**(1), a scrollable window-based terminal emulator that supports a shell.

**Shell Tool** Bring up a **shelltool**(1), an tty-based terminal emulator that supports a shell.

**Graphics Tool** Bring up a **gfxtool**(1), for running graphics programs.

**Console** Bring up a Console window, a **cmdtool** with the **–C** flag, to act as the system console. Since many system messages can be directed to the console, there should always be a console window on the screen.

**Editors**

**Text Editor** Bring up a **textedit**(1), for reading and editing text files.

**Defaults Editor** Bring up a **defaultsedit**(1), for browsing or changing your defaults settings.

**Icon Editor** Bring up a new **iconedit**(1).

**Font Editor** Bring up a **fontedit**(1).

**Tools**

**Mail Tool** Bring up a **mailtool**(1), for reading and sending mail.

**Dbx (Debug) Tool**
Bring up a **dbxtool**(1), a window-based source debugger.

**Performance Meter**
Bring up a **perfmeter**(1) to monitor system performance.

**Clock** Bring up a new **clock**(1).

**Services**

**Redisplay All** Redraw the entire screen. Use this to repair damage done by processes that wrote to the screen without consulting the SunView system.

**Printing** There are two items on this submenu, **Check Printer Queue** and **Print Selected Text**. **Check Printer Queue** displays the printer queue in your console; **Print Selected Text** sends selected text to the standard printer.

**Remote Login** There are two items on this submenu, '**Command Tool**' and '**Shell Tool**'. Each creates a terminal emulator that prompts for a machine name and then starts a shell on that machine.

**Save Layout** Writes out a ~/**.sunview** file that **sunview** can then use when starting up again. An existing ~/**.sunview** file is saved as ~/**.sunview**–.

**Lock Screen** Completely covers the screen with a graphics display, and "locks" the workstation until you type your password. When you "unlock" the workstation, the screen is restored as it was when you locked it. See **lockscreen**(1) for details.

**Exit SunView** Exit from **sunview**, including all windows, and kill processes associated with them. You return to the shell from which you started **sunview**.

You can specify your own SunView menu; see **SunView Defaults** below for details.

**The Frame Menu**

A small set of universal functions are available through the Frame menu. There are also accelerators for some of these functions, described under **Frame Menu Accelerators**, below.

You can invoke the Frame menu when the cursor is over a part of the application that does not provide an application-specific menu, such as the frame header (broad stripe holding the application's name), the border stripes of the window, and the icon.

**Close**

**Open** Toggle the application between closed (iconic) and open state. Icons are placed on the screen according to the icon policy in your defaults database; see **SunView Defaults** below. When a window is closed, its underlying processes continue to run.

**Move** Moves the application window to another spot on the screen. **Move** has a sub-menu with two items: **Unconstrained** and **Constrained**.

**Unconstrained** Move the window both horizontally and vertically.

**Constrained** Moves are either vertical or horizontal, but not both.

Choosing **Move** invokes an **Unconstrained** move.

**Resize** Shrink or stretch the size of a window on the screen. **Resize** has a sub-menu containing:

**Unconstrained** Resize the window both horizontally and vertically.

**Constrained** Resize vertically or horizontally, but not both.

Choosing **Resize** invokes an **Unconstrained** resize.

**UnZoom**

**Zoom** **Zoom** expands a window vertically to the full height of the screen. **UnZoom** undoes this.

**FullScreen** Make a window the full height and width of the screen.

**Front** Bring the window to "the top of the pile." The whole window becomes visible, and hides any window it happens to overlap on the screen.

**Back** Put the window on the "bottom of the pile". The window is hidden by any window which overlaps it.

**Props** Display the property sheet. (Only active for applications that provide a property sheet.)

**Redisplay** Redraw the contents of the window.

**Quit** Notify the application to terminate gracefully. Requires confirmation.

**Frame Menu Accelerators**

Accelerators are provided for some Frame menu functions. You can invoke these functions by pushing a single button in the window's frame header or outer border. See the **SunView Beginner's Guide** for more details.

**Open** Click the LEFT mouse button when the pointer is over the icon.

**Move** Press and hold the MIDDLE mouse button while the pointer is in the frame header or outer border. A bounding box that tracks the mouse is displayed while you hold the button down. When you release the button, the window is redisplayed within the bounding box. If the pointer is near a corner, the move is **Unconstrained**. If it is in the center third of an edge, the move is **Constrained**.

**Resize** Hold the CTRL key and press and hold the MIDDLE mouse button while the pointer is in the frame header or outer border. A bounding box is displayed, and one side or corner tracks the mouse. If the pointer is near a corner when you press the mouse button, the resize is **Unconstrained**; if in the middle third of an edge, the resize is **Constrained**.

**Zoom**
**UnZoom** Hold the CTRL key and click the LEFT mouse button while the pointer is in the frame header or outer border.

**Front** Click the LEFT mouse button while the pointer is on the frame header or outer border.

**Back** Hold the SHIFT key and click the LEFT mouse button while the pointer is on the frame header or outer border.

In addition, you can use two function keys as even faster accelerators. To expose a window that is partially hidden, press the **Front** function key (normally L5) while the pointer is anywhere in that window. Or, if the window is completely exposed, use the **Front** key to hide it. Similarly, to close an open window, press the **Open** key (normally L7) while the pointer is anywhere in that window. If the window is iconic, use the **Open** key to open it.

In applications with multiple windows, you can often adjust the border between two windows up or down, without changing the overall size of the application: hold the CTRL key, press the MIDDLE mouse button over the boundary between the two windows, and adjust the size of the (bounded) subwindow as with **Resize**.

**Startup Processing: The .sunview File**

Unless you override it, **sunview** starts up with a predefined layout of windows. The default layout is specified in the file **/usr/lib/.sunview**. If there is a file called **.sunview** in your home directory, it is used instead. For compatibility with earlier releases, if there is no **.sunview** file in your home directory, but a **.suntools** file instead, the latter file is used.

**SunView Defaults**

SunView allows you to customize the behavior of applications and packages by setting options in a defaults database (one for each user). Use **defaultsedit**(1) to browse and edit your defaults database. Select the "SunView" category to see the following items (and some others):

**Walking_menus** If enabled, the SunView menu, the Frame menu, and many applications will use walking menus. Applications that have not been converted will still use stacking menus. If disabled, applications will use stacking menus. The default value is "Enabled."

**Click_to_Type** If enabled, keyboard input will stay in a window until you click the LEFT or MIDDLE mouse button in another window. If disabled, keyboard input will follow the mouse. The default value is "Disabled."

**Font** You can change the SunView default font by giving the full pathname of the font you want to use. Some alternate fonts are in the directory **/usr/lib/fonts/fixedwidthfonts**. The default font from the SunOS 2.0 release was **/usr/lib/fonts/fixedwidthfonts/screen.r.13**. The default value is null, which has the same effect as specifying **/usr/lib/fonts/fixedwidthfonts/screen.r.11**.

**Rootmenu_filename** You can change the SunView menu by giving the full pathname of a file that specifies your own menu. See **The SunView Menu File** below for details. The default value is null, which gives you the menu found in **/usr/lib/.rootmenu**.

**Icon_gravity** Determine which edge of the screen ("North", "South", "East", or "West") icons will place themselves against. The default value is "North."

**Audible_bell** If enabled, the "bell" command will produce a beep. The default value is "Enabled."

**Visible_bell** If enabled, the "bell" command will cause the screen to flash. The default value is "Enabled."

**Root_Pattern** Used to specify the "pattern" that covers the background. "on" means to use the default desktop gray pattern. "off" means to not use the default desktop gray pattern. "gray" means to use a 50% gray color on color monitors. Anything else is the name of a file produced with **iconedit**(1) which contains an image that is replicated all over the background. The default value is "on."

After you have set the options you want in the "SunView" category, click on the **Save** button in **defaultsedit**; then exit **sunview** and restart it.

Select the "Menu" category to see the following items (and some others):

**Stay_up** If enabled, menus are invoked by pressing and releasing the RIGHT mouse button; the menu appears after you release the RIGHT mouse button. To choose a menu item, point at it, then press and release the RIGHT mouse button a second time. The default value is "False".

**Items_in_column_major**

If enabled, menus that have more than one column are presented in "column major" order (the way **ls**(1V) presents file names). This may make a large menu easier to read. The default value is "False."

After you have set the options you want in the "Menu" category, click on the **Save** button in **defaultsedit**. Any applications you start after saving your changes will be affected by your new choices. For all defaults categories except for "SunView", you do *not* need to exit **sunview** and restart it.

**The SunView Menu File**

The file called **/usr/lib/.rootmenu** contains the specification of the default SunView menu. You can change the SunView menu by creating your own file and giving its name in the **Rootmenu_filename** item in the SunView Defaults.

Lines in the file have the following format: The left side is a menu item to be displayed, and the right side is a command to be executed when that menu item is chosen. You can also include comment lines (beginning with a '#') and blank lines.

The menu item can be a string, or the full pathname of an icon file delimited by angle brackets (unless **Walking_menus** is disabled in the SunView defaults). Strings with embedded blanks must be delimited by double quotes.

There are four reserved-word commands that can appear on the right side.

| | |
|---|---|
| **EXIT** | Exit **sunview** (requires confirmation). |
| **REFRESH** | Redraw the entire screen. |
| **MENU** | This menu item is a pull-right item with a submenu. If a full pathname follows the **MENU** command, the submenu contents are taken from that file. Otherwise, all the lines between a **MENU** command and a matching **END** command are added to the submenu. |
| **END** | Mark the end of a nested submenu. The left side of this line should match the left side of a line with a **MENU** command. |

If the command is not one of these four reserved-word commands, it is treated as a command line and executed. No shell interpretation is done, although you can run a shell as a command.

Here is a menu file that demonstrates some of these features:

| | |
|---|---|
| **Quit** | **EXIT** |
| **"Mail reader"** | **mailtool** |
| **"My tools"** | **MENU /home/me/mytools.menu** |
| **"Click to type"** | **swin -c** |
| **"Follow mouse"** | **swin -m** |
| **"Print selection"** | **sh -c get_selection \| lpr** |
| **"Nested menu"** | **MENU** |
| **"Command Tool"** | **cmdtool** |
| **"Shell Tool"** | **shelltool** |
| **"Nested menu"** | **END** |
| **"Icon menu"** | **MENU** |

```
<images/textedit.icon>   textedit
<images/dbxtool.icon>    dbxtool
"Icon menu"              END
```

**Multiple Screens**

The **sunview** program runs on either a monochrome or color screen. Each screen on a machine with multiple screens may have a separate **sunview** running. The keyboard and mouse input devices can be shared between screens. Using **adjacentscreens**(1) you can set up the pointer to slide from one screen to another when you move it off the edge of a screen.

To set up an instance of **sunview** on two screens:

- Invoke **sunview** on the first display as you normally would. This starts an instance of **sunview** on the default frame buffer (**/dev/fb**).
- In a **shelltool**, run:

  **sunview –d** *device* **&**

  This starts another device. A typical choice might be **/dev/cgone**.
- In that same **shelltool**, run:

  **adjacentscreens /dev/fb –r** *device*

  This sets up the cursor to switch between screens as it crosses the right or left edge of the respective screens.

**Multiple Desktops on the Same Screen**

Machines that support multiple plane groups, such as the Sun-3/110 system, can support independent **sunview** processes on each plane group. They can share keyboard and mouse input in a manner similar to that for multiple screens. To set up two plane groups:

- Start **sunview** in the color plane group by running:

  **sunview –8bit_color_only –toggle_enable**

  This starts **sunview** on the default frame buffer named **/dev/fb**, but limits access to the color plane group.
- In a **shelltool**, run:

  **sunview –d /dev/bwtwo0 –toggle_enable –n &**

  This starts **sunview** in the overlay plane accessed by **/dev/bwtwo0**.
- Run:

  **adjacentscreens –c /dev/fb –l /dev/bwtwo0**

  This sets up the pointer to switch between desktops as it crosses the right or left edge of the respective desktops.

Pre-3.2 applications cannot be run on the **–8bit_color_only** desktop, because they do not write to the overlay plane.

**switcher**(1), another application for switching between desktops, uses some amusing video wipe animation. It can also be used to toggle the enable plane. See **switcher**(1) for details.

### Generic Tool Arguments

Most window-based tools take the following arguments in their command lines:

| FLAG | (LONG FLAG) | ARGUMENTS | NOTES |
|---|---|---|---|
| **-Ww** | **(-width)** | columns | |
| **-Wh** | **(-height)** | lines | |
| **-Ws** | **(-size)** | *x y* | *x* and *y* are in pixels |
| **-Wp** | **(-position)** | *x y* | *x* and *y* are in pixels |
| **-WP** | **(-icon_position)** | *x y* | *x* and *y* are in pixels |
| **-Wl** | **(-label)** | *string* | |
| **-Wi** | **(-iconic)** | | makes the application start iconic (closed) |
| **-Wt** | **(-font)** | *filename* | |
| **-Wn** | **(-no_name_stripe)** | | |
| **-Wf** | **(-foreground_color)** | **red green blue** | 0-255 (no color-full color) |
| **-Wb** | **(-background_color)** | **red green blue** | 0-255 (no color-full color) |
| **-Wg** | **(-set_default_color)** | | (apply color to subwindows too) |
| **-WI** | **(-icon_image)** | *filename* | (for applications with non-default icons) |
| **-WL** | **(-icon_label)** | *string* | (for applications with non-default icons) |
| **-WT** | **(-icon_font)** | *filename* | (for applications with non-default icons) |
| **-WH** | **(-help)** | | print this table |

Each flag option may be specified in either its short form or its long form; the two are completely synonymous.

### SunView Applications

Some of the applications that run in the SunView environment:

**clock**(1), **cmdtool**(1), **dbxtool**(1), **defaultsedit**(1), **fontedit**(1), **gfxtool**(1), **iconedit**(1), **lockscreen**(1), **mailtool**(1), **overview**(1), **perfmeter**(1), **shelltool**(1), **tektool**(1), **textedit**(1), **traffic**(1C)

Some of the utility programs that run in or with the SunView environment:

**adjacentscreens**(1), **clear_functions**(1), **get_selection**(1), **stty_from_defaults**(1), **swin**(1), **switcher**(1), **toolplaces**(1)

## ENVIRONMENT

**DEFAULTS_FILE** The value of this environment variable indicates the file from which SunView defaults are read. When it is undefined, defaults are read from the **.defaults** file in your home directory.

## FILES

**~/.sunview**
**/usr/lib/.sunview**
**/usr/lib/.rootmenu**
**/usr/lib/fonts/fixedwidthfonts/***
**/dev/win***x*
**/dev/ptyp***x*
**/dev/ttyp***x*
**/dev/fb**
**/dev/kbd**
**/dev/mouse**
**/etc/utmp**

**SEE ALSO**

**adjacentscreens**(1), **clear_functions**(1), **clock**(1), **cmdtool**(1), **dbxtool**(1), **defaultsedit**(1), **fontedit**(1), **get_selection**(1), **gfxtool**(1), **iconedit**(1), **lockscreen**(1), **mailtool**(1), **overview**(1), **perfmeter**(1), **screendump**(1), **shelltool**(1), **stty_from_defaults**(1), **swin**(1), **switcher**(1), **tektool**(1), **textedit**(1), **toolplaces**(1), **traffic**(1C), **fbtab**(5), **svdtab**(5)

**BUGS**

Console messages ignore window boundaries unless redirected to a console window. This can disrupt the **sunview** desktop display. The display can be restored using the **Redisplay All** item on the SunView menu. To prevent this, use the **Console** item to start a console window.

With an optical mouse, sometimes the arrow-shaped cursor does not move at start-up; moving the mouse in large circles on its pad normally brings it to life.

**sunview** requires that the **/etc/utmp** file be given read and write permission for all users.

On a color display, colors may "go strange" when the cursor is in certain windows that request a large number of colors.

When running multiple desktops, only one console window can be used.

In **Click-to-type** mode, it is impossible to exit from **sunview** by typing CTRL-D CTRL-Q.

## NAME

sv_acquire, sv_release – change owner, group, permissions of window devices

## SYNOPSIS

**sv_acquire** [ *startwin* [ *nwins* [ *nbackground* ]]]

**sv_release**

## DESCRIPTION

**sv_acquire** changes the owner, group and permissions of window system devices to the user ID, group ID, and the permissions specified in **/etc/svdtab** (see **svdtab**(5)), if there is an entry in that file. If no entry exists in **/etc/svdtab**, no changes will be made.

**sv_release** returns the owner, and group to **root** and **wheel**. The permissions are set as specified in **/etc/svdtab**.

*startwin* indicates the first **win** to change, *nwins* indicates the number of windows to change, *nbackgound* indicates the number of **win** devices to change in the backgound.

This is a setuid root program spawned by the window system (for example, **sunview**(1)). **sv_acquire** exits if the user does not own console devices listed in **/etc/fbtab** (see **fbtab**(5)). **sv_acquire** exits with 0 on success and –1 on failure.

## EXAMPLES

In the following example, **sv_acquire** sets the user ID and the group ID of **/dev/win0** through **/dev/win15** in the foreground and spawns a background process to set **/dev/win16** through **/dev/win256** and exit with error status.

**sv_acquire 0 256 240**

The purpose of spawning a background process is to allow the caller to avoid waiting for all **win** devices to be changed. Note: a diskless 3/50 sets approximately 16 wins per second.

## SEE ALSO

**sunview**(1), **fbtab**(5), **svdtab**(5)

## NAME

swin – set or get SunView user input options

## SYNOPSIS

**swin** [ **–cghm** ] [ **–r** *event value shift_state* ] [ **–s** *event value shift_state* ] [ **–t** *seconds* ]

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

The **swin** (set window; analogous to **stty**(1V)) command lets you change some of the input behavior of your SunView environment. By default, your keyboard input follows your pointer. This means that in order to type to a window you position the pointer over the window. This is called *keyboard-follows-mouse* mode.

You can specify that the keyboard input continues to go to the same window, regardless of the pointer position, until you take some specific action, like clicking the mouse. When this is done, you can roam around the screen with the pointer and not change the window to which keyboard input is directed. Running SunView like this is said to be operating in *click-to-type* mode.

When running in click-to-type mode, one user action *sets* the type-in point in the window that you want to receive keyboard input. The default user action to do this is the clicking of the LEFT mouse button while positioning the pointer over the new type-in point. This user action can be changed.

Another user action *restores* the previous type-in point in the window that you want to receive keyboard input. The default user action to do this is the clicking of the MIDDLE mouse button while positioning the pointer over the window. This user action can be changed.

## OPTIONS

**–c** Turn on click-to-type mode using the default user actions: the LEFT mouse button sets the type-in point and the MIDDLE mouse button restores the type-in point. You can use the **defaultsedit**(1) program to set click-to-type on permanently; see the **Click_to_Type** option of **sunview**(1).

**–g** Get the state of the user input options controlled by **swin**. If no arguments are supplied to **swin** then **–g** is implied.

**–h** Print out a help message that briefly describes the options to **swin**.

**–m** Run in keyboard-follows-mouse mode.

**–s** *event value shift_state*

Set the user action that sets the type-in point and sets the keyboard input window. The *event* identifies the particular user action and is one of:

LOC_WINENTER
: pointer entering a window

MS_LEFT
: LEFT mouse button

MS_MIDDLE
: MIDDLE mouse button

MS_RIGHT
: RIGHT mouse button

*decimal_number*

place the decimal number of a firm event here; see list of events in **<sundev/vuid_event.h>** (avoid function keys, normally unused control-ASCII characters are OK, normally unused SHIFT keys are OK).

*value* identifies the transition of the *event* and is one of:

ENTER the pointer entering a window (use with LOC_WINENTER)

DOWN the button associated with *event* went down

UP the button associated with *event* went up (avoid this)

The *shift_state* identifies the state of the SHIFT keys at the time of the *event/value* pair in order for that pair to be used to control the keyboard input window. The *shift_state* is one of:

SHIFT_DONT_CARE
Ignore the state of the SHIFT keys

SHIFT_ALL_UP
All the SHIFT keys must be up

SHIFT_LEFT
The left SHIFT key must be down (not the key labeled LEFT)

SHIFT_RIGHT
the right SHIFT key must be down (not the key labeled RIGHT)

SHIFT_LEFTCTRL
the left CTRL key must be down

SHIFT_RIGHTCTRL
the right CTRL key must be down

**–r** *event value shift_state*

Set the user action that restores the type-in point and sets the keyboard input window. This user action is swallowed so that the application that owns the window does not see it. However, if the window already has keyboard input or if the window refuses keyboard input then this user action is passed on through to the application. The parameters to this command are like those for **–s**. The following example shows modifying the default click-to-type user actions so that a SHIFT left is required for the restore user event:

**example% swin –c –r MS_MIDDLE DOWN SHIFT_LEFT**

**–t** *seconds*

SunView synchronizes input so that it does not hand out the next user action until the application fielding the current user action finishes its processing. This allows type-ahead and mouse-ahead. If an application does not finish processing within a given length of time (process virtual time; not wall clock time), the next user action is handed out anyway. This avoids any one application from hanging the workstation. The **–t** command sets this time limit. A *seconds* value of 0 tells SunView to run unsynchronized; beware of race conditions in this mode. The default seconds value is 2 and the –c command makes it 10 seconds.

**SEE ALSO**

**defaultsedit**(1), **stty**(1V), **sunview**(1)

*SunView User's Guide*

DIAGNOSTICS

**swin not passed parent window in environment**

**swin** does not work unless SunView is started already.

BUGS

**swin** gets you no help in preventing you from specifying **–r** or **–s** parameters that are not sensible.

## NAME

switcher – switch attention between multiple SunView desktops on the same physical screen

## SYNOPSIS

**switcher** [ **–d** *frame-buffer* ] [ **–s n | l | r | i | o | f** ] [ **–m** *x y* ] [ **–n** ] [ **–e 0 | 1** ]

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**switcher** is used as an alternative to **adjacentscreens**(1) for getting between desktops on the Sun-3/110. Clicking the switcher icon gets you to another desktop using some amusing video-wipe animation. When using walking menus, a menu is available to invoke the switch as well. **switcher** can also be used to simply set the enable plane to 0 or 1 should it get out of wack.

## OPTIONS

**–d** *frame-buffer*

The *frame buffer* is a frame buffer device name, such as **/dev/fb**, **/dev/cgfour** or **/dev/bwtwo0**, on which the desktop that you want to get to resides. This name is the same one supplied to **sunview**. The **–d** flag is optional; if not specified, the default device is **/dev/fb**

**–s n | l | r | i | o | f**

The **–s** flag specifies the type of animation used when switching: **n** (now), **l** (left wipe), **r** (right wipe), **i** (tunnel in), **o** (tunnel out), or **f** (fade). The **–s** flag is optional because if not specified, the default animation is to switch immediately. **n** (now) mode.

**–m***x y* The **–m** indicates what the mouse position should be on the destination desktop after the switch. An (*x y*) value-pair of (–1 –1) says to use the position of the mouse on the desktop at the time of the switch as the mouse position on the destination desktop. The **–m** flag is optional; if not specified, the default is (–1 –1).

**–n** The **–n** flag means no switcher icon is wanted so do the switch right now and exit **switcher** after the switch. This is handy if you want to switch from a root menu command.

**–e 0 | 1** The **–e** flag causes the overlay enable plane of the device specified with the **-d** flag to be set to either 0 (show color) or 1 (show black and white). **switcher** run with this option has nothing to do with SunView, only the enable plane is set.

## EXAMPLE

A common multiple desktop configuration for the Sun-3/110 is one monochrome and one color desktop. You could set up an instance of **sunview**(1) on each plane group in the following ways:

1. Invoke **sunview** in the color plane group by running:

   **example% sunview –8bit_color_only –toggle_enable**

   This starts **sunview** on the default frame buffer named **/dev/fb** but limits access to the color plane group.

2. In a **shelltool**(1), run:

   **example% sunview –d /dev/bwtwo0 –toggle_enable &**

   This starts **sunview** in the overlay plane that is accessed by **/dev/bwtwo0**.

3. In a **shelltool** on the original desktop run:

**example% switcher –d /dev/bwtwo0 –s i &**

Clicking on the switcher icon when it is visible moves you to the **/dev/bwtwo0** desktop.

4. In a **shelltool** on the **/dev/bwtwo0** desktop run:

**example% switcher –s o &**

Clicking on the switcher icon when it is visible moves you back to the **/dev/fb** desktop.

**FILES**

**/usr/bin/switcher**
**/dev/bwtwo0**
**/dev/fb**
**/dev/cgfour**

**SEE ALSO**

**adjacentscreens**(1), **shelltool**(1), **sunview**(1)

**NAME**

symorder – rearrange a list of symbols

**SYNOPSIS**

**symorder** [ **–s** ] *orderlist symbolfile*

**DESCRIPTION**

*sysmfile* is a file containing symbols to be found in objectfile, 1 symbol per line.

*objectfile* is updated in place to put the requested symbols first in the symbol table, in the order specified. This is done by swapping the old symbols in the required spots with the new ones. If all of the order symbols are not found, an error is generated.

This program was specifically designed to cut down on the overhead of getting symbols from **/vmunix**.

**Sun386i DESCRIPTION**

Symbols specified on the command line are moved to the beginning of the output symbol table, not swapped. Therefore, the symbols specified on the command line will appear in order at the beginning of the output symbol table, followed by the original symbol table with the gaps created by the moved symbols closed.

**OPTIONS**

**–s** Work silently, that is, display nothing except error messages. This is useful for checking the error status.

**FILES**

**/vmunix**

**SEE ALSO**

**nlist**(3V)

**NAME**

sync – update the super block; force changed blocks to the disk

**SYNOPSIS**

**sync**

**DESCRIPTION**

**sync** forces any information on its way to the disk to be written out immediately. **sync** can be called to ensure that all disk writes are completed before the processor is halted abnormally.

**SEE ALSO**

**cron**(8), **fsck**(8), **halt**(8), **reboot**(8)

## NAME

sysex – start the system exerciser

## SYNOPSIS

**sysex**

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

**sysex** is the system exerciser for Sun386i systems.

This program is designed to run under the SunView environment, but a dumb terminal interface is provided. The program tests subsystems of a Sun386i system, printing information to the console and log files. Most commands are accessible by way of buttons, toggle switches, or menus. A startup file ~/**.sysexrc**, can be created by experienced users to set runtime parameters.

## USAGE

### Subwindows

| | |
|---|---|
| Control Panel | Tells which version of the exerciser is running. User controls **sysex** executions through the toggles, buttons, and sliders in this panel. |
| Perfmon window | Graphically displays system statics. The standard SunView **perfmeter**(1). |
| Console window | Displays the system messages and **sysex** error messages. |
| Status Window | Displays pass counts and error counts for all tests that are currently selected. Also displays system pass count, and total system errors. Elapsed time signifies time since start button is pressed. |

### Load Sliders

These slide bars allow the user to modulate the load on the system.

I/O-CPU Load

Moving this slider changes the balance between I/O-intensive and compute-intensive tests that are running.

SYSTEM LOAD

Moving this slider increases and decreases the system activity generated by **sysex**.

### Test Toggles

Each test selection on the control panel is selectable by moving the cursor over to the toggle and pressing the left mouse button.

The tests are displayed by device groups in the control panel. A test is enabled when a check mark is seen in the box. Clicking left on the group label acts as a group enable/disable for all tests in that device group. Currently there are tests for physical memory and virtual memory, fixed disk, diskette, Ethernet, and color frame buffer.

### Command Buttons

Command buttons exist for the following commands:

| | |
|---|---|
| **Quit Sysex** | Stop all current tests and exit the exerciser. All logs will be saved. |
| **Log Files** | Display menu for choosing a log to view, reset, or print. |
| **Options** | Display window through which ~/**.sysexrc** parameters can be modified. |
| **Print Screen** | Take screendump of the current screen. |

**Start Tests** Start all test from pass0 that have been selected. Resets pass count. Toggles to Stop Tests. Begin elapsed time count.

**Stop Tests** Stop all tests that are running. Toggles to Start Tests.

**Pause** Pause tests by issuing SIGSTP.

**Continue** Continue testing from the stopped state without resetting pass count. Toggles to Pause,leaves pass counts intact. Continue elapsed time count if Continued from a Pause.

**Logs**

When the user selects the DISPLAY LOGS button and chooses from the log menu, a scrollable pop-up window displays the log, which is one of **/var/sysex/sysex.info**, **/var/sysex/sysex.error**, or **/etc/adm/messages**. Logs contain messages classified as INFO, WARNING, ERROR, or FATAL. The INFO file contains all messages; the ERROR file contains only error and fatal messages.

**Variables**

**sysex** has several variables than can be set in the **~/.sysexrc** file. Some of the variables pertain to only one test and others are global to all tests. Clicking **Done** saves changes to the **~/.sysexrc** file. Clicking **Cancel** leaves options unchanged.

**verbose** Display messages about what is currently taking place.

**verify** Run through a cursory pass of tests to see all subsystems present.

**run_on_err**
Halt subsystem testing when an error occurs.

**sysex_halt_on_err**
Stop **sysex** if an error occurs in any subsystem.

**core** Create core dump in */var/sysex*.

**single_pass**
Run one pass of each selected device test.

For the expert user, more commands are available by clicking the manufacturing cycle to Enabled. This displays the following options:

**fdc_wait** Variable wait time between executions of the diskette test.

**vmem_wait**
Variable delay between successive executions of the virtual memory test.

**debug** Display all messages to aid analysis of problem systems.

**check_eeprom**
Read **NVRAM** configuration information and display values.

**intervention**
Turn on or off the confirmer for all destructive tests, or for tests requiring media.

**FILES**

**/usr/sysex/sysex**
**/var/sysex/core**
**/var/sysex/sysex.info**
**/var/sysex/sysex.error**

**NAME**

syswait – execute a command, suspending termination until user input

**SYNOPSIS**

**syswait** *message command*

**AVAILABILITY**

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

**DESCRIPTION**

**syswait** executes a specified command, suspending termination until the user types any character. *message* is the message prompting the user to type a character to terminate the command. *command* is the command to be executed.

**EXAMPLE**

The following example invokes a cmdtool and executes '**ls *.c**', but waits for the user to type a character before terminating the **ls** and closing the cmdtool window.

**cmdtool syswait "Press any key to quit..." "ls *.c" &**

## NAME

tabs – set tab stops on a terminal

## SYNOPSIS

**/usr/5bin/tabs** [ *tabspec* ] [ **–T***type* ] [ **+m***n* ]

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**tabs** can be used to specify TAB stops on terminals that support remotely-settable hardware TAB characters. TAB stops are set according to the *tabspec* option, as described below, and previous settings are erased.

Four types of tab specification are accepted for *tabspec*. They are:

*–code* Set the TAB stops according to the canned TAB setting specified by *code*, as given by **fspec**(5).

*–n* Set the TAB stops at intervals of *n* columns, that is, at 1+*n*, 1+2**n*, and so on, as per the *–n* specification as given by **fspec**(5).

*n1,n2,...*
Set the TAB stops at positions *n1*, *n2*, and so on, as per the *n1,n2,...* specification as given by **fspec**(5).

*––file* Read the first line of the file specified by *file*, searching for a format specification as given by **fspec**(5). If this line contains a format specification, set the TAB stops accordingly, otherwise set them to every 8 columns. This type of specification may be used to make sure that a file containing a TAB specification is displayed with correct TAB settings. For example, it can be used with the **pr**(1V) command:

**tabs ––file; pr file**

If no *tabspec* is given, the default value is **–8**, the standard default TAB setting. The lowest column number is 1. Note: for **tabs**, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, (such as the DASI 300, DASI 300s, and DASI 450). TAB and margin setting is performed by echoing to the proper sequences to the standard output.

## OPTIONS

**–T***type* **tabs** usually needs to know the type of terminal in order to set TAB characters, and always needs to know to set margins. *type* is a name listed in **term**(5). If no **–T** flag is supplied, **tabs** uses the value of the environment variable **TERM**. If **TERM** is not defined in the environment (see **environ**(5V)), **tabs** tries a default sequence that will work for many terminals.

**+m***n* The margin argument may be used for some terminals. It moves all TAB stops over *n* columns by making column *n*+1 the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

## SEE ALSO

**pr**(1V), **tput**(1V), **environ**(5V), **fspec**(5), **term**(5), **terminfo**(5V)

**BUGS**

There is no consistency between different terminals regarding ways of clearing tabs and setting the left margin.

**tabs** clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

## NAME

tail – display the last part of a file

## SYNOPSIS

**tail** + | –*number* [ **lbc** ] [ **f** ] [ *filename* ]
**tail** + | –*number* [ **l** ] [ **rf** ] [ *filename* ]

## DESCRIPTION

**tail** copies *filename* to the standard output beginning at a designated place. If no file is named, the standard input is used.

## OPTIONS

Options are all jammed together, not specified separately with their own '–' signs.

+*number*

Begin copying at distance *number* from the beginning of the file. *number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines. If *number* is not specified, the value 10 is used.

–*number*

Begin copying at distance *number* from the end of the file. *number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines. If *number* is not specified, the value 10 is used.

**l** *number* is counted in units of lines.

**b** *number* is counted in units of blocks.

**c** *number* is counted in units of characters.

**r** Copy lines from the end of the file in reverse order. The default for option **r** is to print the entire file in reverse order. *number* is the count of lines from the end of the file regardless of sign. Option **r** may not be used with **b** or **c**.

**f** If the input file is not a pipe, do not terminate after the line of the input file has been copied, but enter an endless loop, sleeping for a second and then attempting to read and copy further records from the input file. This option may be used to monitor the growth of a file that is being written by some other process. For example, the command:

**tail –f fred**

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time **tail** is initiated and killed. As another example, the command:

**tail –15cf fred**

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time **tail** is initiated and killed.

## SEE ALSO

**dd**(1)

## BUGS

Data for a tail relative to the end of the file is stored in a buffer, and thus is limited in size.

Various kinds of anomalous behavior may happen with character special files.

**NAME**

talk – talk to another user

**SYNOPSIS**

**talk** *username* [ *ttyname* ]

**DESCRIPTION**

**talk** is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on your own machine, then *username* is just the person's login name. If you wish to talk to a user on another host, then *username* is one of the following forms :

*host*!*user*
*host*.*user*
*host*:*user*
*user*@*host*

though *user*@*host* is perhaps preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, **talk** sends the message:

**Message from TalkDaemon@***his_machine* **at***time*...
**talk: connection requested by** *your_name*@*your_machine*.
**talk: respond with: talk** *your_name*@*your_machine*

to the user you wish to talk to. At this point, the recipient of the message should reply by typing:

**example% talk** *your_name*@*your_machine*

It does not matter from which machine the recipient replies, as long as their login name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing CTRL–**L** redraws the screen, while your erase, kill, and word kill characters will work in **talk** as normal. To exit, just type your interrupt character; **talk** then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the **mesg** command. At the outset talking is allowed. Certain commands, in particular **nroff**(1) and **pr**(1V) disallow messages in order to prevent messy output.

**FILES**

| | |
|---|---|
| **/etc/hosts** | to find the recipient's machine |
| **/etc/utmp** | to find the recipient's tty |

**SEE ALSO**

**mail**(1), **mesg**(1), **nroff**(1), **pr**(1V), **who**(1), **write**(1), **talkd**(8C)

## NAME

tar – create tape archives, and add or extract files

## SYNOPSIS

**tar** [ – ] **c|r|t|u|x** [ **bBefFhilmopvwX014578** ] [ *tarfile* ] [ *blocksize* ] [ *exclude-file* ] [ **–I** *include-file* ] *filename1 filename2 ...* **–C** *directory filenameN ...*

## DESCRIPTION

**tar** archives and extracts multiple files onto a single **tar**, file archive, called a *tarfile*. A tarfile is usually a magnetic tape, but it can be any file. **tar**'s actions are controlled by the first argument, the *key*, a string of characters containing exactly one function letter from the set **crtux**, and one or more of the optional function modifiers listed below. Other arguments to **tar** are file or directory names that specify which files to archive or extract. In all cases, the appearance of a directory name refers recursively to the files and subdirectories of that directory.

## FUNCTION LETTERS

**c** Create a new *tarfile* and write the named files onto it.

**r** Write the named files on the end of the *tarfile*. Note: this option *does not work* with quarter-inch archive tapes.

**t** List the table of contents of the *tarfile*.

**u** Add the named files to the *tarfile* if they are not there or if they have been modified since they were last archived. Note: this option *does not work* with quarter-inch archive tapes.

**x** Extract the named files from the *tarfile*. If a named file matches a directory with contents written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *filename* arguments are given, all files in the archive are extracted. Note: if multiple entries specifying the same file are on the tape, the last one overwrites all earlier versions.

## FUNCTION MODIFIERS

**b** Use the next argument as the blocking factor for tape records. The default blocking factor is 20 blocks. The block size is determined automatically when reading tapes (key letters **x** and **t**). This determination of the blocking factor may be fooled when reading from a pipe or a socket (see the **B** key letter below). The maximum blocking factor is determined only by the amount of memory available to **tar** when it is run. Larger blocking factors result in better throughput, longer blocks on nine-track tapes, and better media utilization.

**B** Force **tar** to perform multiple reads (if necessary) so as to read exactly enough bytes to fill a block. This option exists so that **tar** can work across the Ethernet, since pipes and sockets return partial blocks even when more data is coming.

**e** If any unexpected errors occur **tar** will exit immediately with a positive exit status.

**f** Use the next argument as the name of the *tarfile*. If **f** is omitted, use the device indicated by the **TAPE** environment variable, if set. Otherwise, use **/dev/rmt8** by default. If *tarfile* is given as '–', **tar** writes to the standard output or reads from the standard input, whichever is appropriate. Thus, **tar** can be used as the head or tail of a filter chain. **tar** can also be used to copy hierarchies with the command:

**example% cd fromdir; tar cf – . | (cd todir; tar xfBp –)**

**F** With one **F** argument specified, exclude all directories named SCCS from *tarfile*. With two arguments **FF**, exclude all directories named SCCS, all files with **.o** as their suffix, and all files named **errs**, **core**, and **a.out**.

**h** Follow symbolic links as if they were normal files or directories. Normally, **tar** does not follow symbolic links.

**i** Ignore directory checksum errors.

**l** Display error messages if all links to archived files cannot be resolved. If **l** is not used, no error messages are printed.

**m** Do not extract modification times of extracted files. The modification time will be the time of extraction.

**o** Suppress information specifying owner and modes of directories which **tar** normally places in the archive. Such information makes former versions of **tar** generate an error message like:

**filename/**: *cannot create*

when they encounter it.

**p** Restore the named files to their original modes, ignoring the present **umask**(2V). SetUID and sticky information are also extracted if you are the super-user. This option is only useful with the **x** key letter.

**v** Verbose. Normally **tar** does its work silently; this option displays the name of each file **tar** treats, preceded by the function letter. When used with the **t** function, **v** displays the *tarfile* entries in a form similar to '**ls –l**'.

**w** Wait for user confirmation before taking the specified action. If you use **w**, **tar** displays the action to be taken followed by the file name, and then waits for a **y** response to proceed. No action is taken on the named file if you type anything other than a line beginning with **y**.

**X** Use the next argument as a file containing a list of named files (or directories) to be excluded from the *tarfile* when using the key letters **c**, **x**, or **t**. Multiple **X** arguments may be used, with one *exclude file* per argument.

**014578** Select an alternate drive on which the tape is mounted. The numbers **2**, **3**, **6**, and **9** do not specify valid drives. The default is **/dev/rmt8**.

If a file name is preceded by –**I** then the filename is opened. A list filenames, one per line, is treated as if each appeared separately on the command line. Be careful of trailing white space in both include and exclude file lists.

In the case where excluded files (see **X** option) also exist, excluded files take precedence over all included files. So, if a file is specified in both the include and exclude files (or on the command line), it will be excluded.

If a file name is preceded by –**C** in a **c** (create) or **r** (replace) operation, **tar** will perform a **chdir** (see **csh**(1)) to that file name. This allows multiple directories not related by a close common parent to be archived using short relative path names. See EXAMPLES below.

Note: the –**C** option only applies to *one* following directory name and *one* following file name.

## EXAMPLES

To archive files from **/usr/include** and from **/etc**, one might use:

**example% tar c –C /usr include –C /etc .**

If you get a table of contents from the resulting *tarfile*, you will see something like:

**include/**
**include/a.out.h**
*and all the other files in* **/usr/include** ...
**/chown**
*and all the other files in* **/etc**

Here is a simple example using **tar** to create an archive of your home directory on a tape mounted on drive **/dev/rmt0**:

```
example% cd
example% tar cvf /dev/rmt0 .
messages from tar
```

The **c** option means create the archive; the **v** option makes **tar** tell you what it is doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (**/dev/rmt0** in this example).

Now you can read the table of contents from the archive like this:

```
example% tar tvf /dev/rmt0          display table of contents of the archive
(access user-id/group-id    size    mod. date              filename)
rw-r--r-- 1677/40           2123    Nov 7 18:15:1985       ./archive/test.c
...
example%
```

You can extract files from the archive like this:

```
example% tar xvf /dev/rmt0          extract files from the archive
messages from tar
example%
```

If there are multiple archive files on a tape, each is separated from the following one by an EOF marker. **tar** does not read the EOF mark on the tape after it finishes reading an archive file because **tar** looks for a special header to decide when it has reached the end of the archive. Now if you try to use **tar** to read the next archive file from the tape, **tar** does not know enough to skip over the EOF mark and tries to read the EOF mark as an archive instead. The result of this is an error message from **tar** to the effect:

```
tar: blocksize=0
```

This means that to read another archive from the tape, you must skip over the EOF marker before starting another **tar** command. You can accomplish this using the **mt**(1) command, as shown in the example below. Assume that you are reading from **/dev/nrmt0**.

```
example% tar xvfp /dev/nrmt0        read first archive from tape
messages from tar
example% mt fsf 1                   skip over the end-of-file marker
example% tar xvfp /dev/nrmt0        read second archive from tape
messages from tar
example%
```

Finally, here is an example using **tar** to transfer files across the Ethernet. First, here is how to archive files from the local machine (**example**) to a tape on a remote system (**host**):

```
example% tar cvfb - 20 filenames | rsh host dd of=/dev/rmt0 obs=20b
messages from tar
example%
```

In the example above, we are *creating* a *tarfile* with the **c** key letter, asking for *verbose* output from **tar** with the **v** option, specifying the name of the output *tarfile* using the **f** option (the standard output is where the *tarfile* appears, as indicated by the '–' sign), and specifying the blocksize (20) with the **b** option. If you want to change the blocksize, you must change the blocksize arguments both on the **tar** command *and* on the **dd** command.

Now, here is how to use **tar** to get files from a tape on the remote system back to the local system:

```
example% rsh -n host dd if=/dev/rmt0 bs=20b | tar xvBfb - 20 filenames
messages from tar
example%
```

In the example above, we are *extracting* from the *tarfile* with the **x** key letter, asking for *verbose output from* **tar** with the **v** option, telling **tar** it is reading from a pipe with the **B** option, specifying the name of the input *tarfile* using the **f** option (the standard input is where the *tarfile* appears, as indicated by the '–' sign), and specifying the blocksize (20) with the **b** option.

**FILES**

| | |
|---|---|
| **/dev/rmt?** | half-inch magnetic tape interface |
| **/dev/rst?** | SCSI tape interface |
| **/tmp/tar*** | |

**ENVIRONMENT**

**TAPE** If specified, in the environment, the value of **TAPE** indicates the default tape device.

**SEE ALSO**

**cpio**(1), **csh**(1), **mt**(1), **umask**(2V), **tar**(5), **dump**(8), **restore**(8)

**BUGS**

Neither the **r** option nor the **u** option can be used with quarter-inch archive tapes, since these tape drives cannot backspace.

There is no way to ask for the *n*th occurrence of a file.

Tape errors are handled ungracefully.

The **u** option can be slow.

There is no way selectively to follow symbolic links.

When extracting tapes created with the **r** or **u** options, directory modification times may not be set correctly.

Files with names longer than 100 characters cannot be processed.

Filename substitution wildcards do not work for extracting files from the archive. To get around this, use a command of the form:

**tar xvf... /dev/rst0 'tar tf... /dev/rst0 | grep '***pattern* **''**

## NAME

tbl – format tables for nroff or troff

## SYNOPSIS

**tbl** [ **–ms** ] [ *filename* ] ...

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**tbl** is a preprocessor for formatting tables for **nroff**(1) or **troff**(1). The input *filename*s are copied to the standard output, except that lines between **.TS** and **.TE** command lines are assumed to describe tables and are reformatted. Details are given in *Formatting Documents*.

If no arguments are given, **tbl** reads the standard input, so **tbl** may be used as a filter. When **tbl** is used with **eqn**(1) or **neqn** the **tbl** command should be first, to minimize the volume of data passed through pipes.

## OPTIONS

**–ms** Copy the **–ms** macro package to the front of the output file.

## EXAMPLE

As an example, letting \t represent a TAB (which should be typed as a genuine TAB) the input

```
.TS
c s s
c c s
c c c
l n n.
Household\tPopulation
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

| Town | Household Population | |
|---|---|---|
| | Households | |
| | Number | Size |
| Bedminster | 789 | 3.26 |
| Bernards Twp. | 3087 | 3.74 |
| Bernardsville | 2018 | 3.30 |
| Bound Brook | 3425 | 3.04 |
| Branchburg | 1644 | 3.49 |
| Bridgewater | 7897 | 3.81 |
| Far Hills | 240 | 3.19 |

**SEE ALSO**

**eqn**(1), **nroff**(1), **troff**(1)

*Formatting Documents*

**NAME**

tcopy – copy a magnetic tape

**SYNOPSIS**

**tcopy** *source* [ *destination* ]

**DESCRIPTION**

**tcopy** copies the magnetic tape mounted on the tape drive specified by the *source* argument. The only assumption made about the contents of a tape is that there are two tape marks at the end.

When only a source drive is specified, **tcopy** scans the tape, and displays information about the sizes of records and tape files. If a destination is specified, **tcopy** makes a copies the source tape onto the *destination* tape, with blocking preserved. As it copies, **tcopy** produces the same output as it does when only scanning a tape.

**SEE ALSO**

**mt**(1)

## NAME

tcov – construct test coverage analysis and statement-by-statement profile

## SYNOPSIS

**tcov** [ **–a** ] [ *–n* ] *srcfile...*

## AVAILABILITY

This command is available on Sun-3 and Sun-4 systems only.

## DESCRIPTION

**tcov** produces a test coverage analysis and statement-by-statement profile of a C or FORTRAN program. When a program in a file named *file*.**c** or *file*.**f** is compiled with the **–a** option, a corresponding *file*.**d** file is created. Each time the program is executed, test coverage information is accumulated in *file*.**d**.

**tcov** takes source files as arguments. It reads the corresponding *file*.**d** file and produces an annotated listing of the program with coverage data in *file*.**tcov**. Each straight-line segment of code (or each line if the **–a** option to **tcov** is specified) is prefixed with the number of times it has been executed; lines which have not been executed are prefixed with #####.

Note: the profile produced includes only the number of times each statement was executed, not execution times; to obtain times for routines use **gprof**(1) or **prof**(1).

## OPTIONS

**–a** Display an execution count for each statement; if **–a** is not specified, an execution count is displayed only for the first statement of each straight-line segment of code.

*–n* Display table of the line numbers of the *n* most frequently executed statements and their execution counts.

## EXAMPLES

The command:

**example% cc -a -o prog prog.c**

compiles with the **-a** option — produces **prog.d**

The command: **example% prog**

executes the program '–' accumulates data in **prog.d**

The command:

**example% tcov prog.c** produces an annotated listing in file **prog.tcov**

## FILES

| | |
|---|---|
| **file.c** | input C program file |
| **file.f** | input FORTRAN program file |
| **file.d** | input test coverage data file |
| **file.tcov** | output test coverage analysis listing file |
| **/usr/lib/bb_link.o** | entry and exit routines for test coverage analysis |

## SEE ALSO

**cc**(1V), **gprof**(1), **prof**(1), **exit**(2V)

## DIAGNOSTICS

**premature end of file**

Issued for routines containing no statements.

## BUGS

The analyzed program must call **exit**(2V) or return normally for the coverage information to be saved in the *.d* file.

**NAME**

tee – replicate the standard output

**SYNOPSIS**

**tee** [ **–ai** ] [ *filename* ] ...

**DESCRIPTION**

**tee** transcribes the standard input to the standard output and makes copies in the *filename*s.

**OPTIONS**

**–a** Append the output to the *filename*s rather than overwriting them.

**–i** Ignore interrupts.

## NAME

tektool – SunView Tektronix 4014 terminal-emulator window

## SYNOPSIS

**tektool** [ **–s**[ **lcdeg**[ **ce** ]**m**[ **12** ] ] [ **–**[ **cr** ] *command-line* ] [ **–f***fontdir* ]

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**tektool** emulates a Tektronix 4014 terminal with the enhanced graphics module. It does this in much the same way as shelltool (see **shelltool**(1)) emulates a regular glass tty. When **tektool** is invoked, a command (usually a shell) is started up, its output and input are connected to the emulator, and a new window is formed. The default window is the entire screen. When the emulator is running, keys TF(1) through TF(3), (usually function keys F1-F3 (see **kbd**(4S)) have special meaning.

TF (1) Unshifted, this is the 4014 **PAGE** button. Shifted, this is the 4014 **RESET** button.

TF (2) Copy screen. The raster image (**<rasterfile.h>**) of the 4014 screen is piped to a command found in the **TEKCOPY** environment variable. If **TEKCOPY** is not found in the environment, '**lpr –v**' is used. The copy button is unaffected by window manipulations, and will transmit the contents of the 4014 screen only.

TF (3) Release page full condition.

These functions are also available through the tool menu. When in graphics input (GIN) mode and the 4014 crosshairs are visible, the left hand mouse button may be used as the space bar to terminate GIN mode.

## OPTIONS

**–s** Specifies the Tektronix 4014 strap options with the following modifiers:

- l Received LINEFEED characters also generate RETURN characters.
- c Received RETURN characters also generate LINEFEED characters.
- d Received DELETE characters are used as low order Y axis ( LOY ) addresses.
- e Echo keyboard input.
- g Graphic input mode (GIN) terminator specification. If this strap is followed by a **c**, GIN mode data is terminated by a RETURN character. If it is followed by a **e**, GIN mode data is terminated by a RETURN character followed by an EOT character. If this strap is not present, no characters are sent after GIN mode data.
- m Page full control specification. If this strap is followed by a **1**, **tektool** will stop accepting tty input when a LINEFEED character is done past the last line in margin 1. This is the 4014 page full condition. The page full condition it released by a **PAGE** or a **RELEASE** or any ASCII keyboard input. If this strap is followed by a **2**, the page full condition happens at the end of margin 2. If this strap is not present, the page full condition never occurs.

If the **–s** option is not given, the environment is searched for the **TEKSTRAPS** variable which provides the modifiers. The straps may also be set by the property sheet available by selecting the **PROPS** menu item. If no straps are specified the **d** strap is assumed.

**–c** *command-line* Take terminal emulator input from a shell which in turn runs the *command-line* following the **–c** option.

**–f** *fontdir* Look for fonts in the directory specified by *fontdir*. The fonts must be called **tekfont**0 through **tekfont**3. Fonts must be in **vfont**(5) format. If this option is not given, the font directory is obtained from the TEKFONTS environment variable (if it exists). If no font directory is specified, **/usr/lib/fonts/tekfonts** is used.

**–r** *command-line* Run *command-line* to provide input to the terminal emulator. This must be the last option, since the remainder of the arguments are used by the command.

**FILES**

**/usr/lib/fonts/tekfonts/tekfont[0-3]**

**SEE ALSO**

**shelltool**(1), **sunview**(1), **kbd**(4S), **vfont**(5)

*Installing SunOS 4.1*

**BUGS**

Special point plot mode is not supported.

Z-axis stuff, except for defocusing, is not supported.

Defocused alpha characters are not supported.

**DIAGNOSTICS**

**copy command failed** The copy command in the **TEKCOPY** environment variable or in the property sheet could not be executed.

**WARNINGS**

Like all 4014 emulators, this does not duplicate every nuance of the 4014. For instance, certain programs redraw stuff already on the screen in order to highlight things with the storage flash. This will not work here. Also, even though the emulator supports the full 4096 point addressing of the 4014, it cannot display this on the screen. All points will be rounded to the nearest available pixel. This may cause some funny effects.

The **tektool** window may be treated just like other windows; it can be overlaid, moved, reshaped etc. However, when the window is reshaped, the contents will not scale.

## NAME

telnet – user interface to a remote system using the TELNET protocol

## SYNOPSIS

**telnet** [ *host* [ *port* ] ]

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**telnet** communicates with another host using the TELNET protocol. If **telnet** is invoked without arguments, it enters command mode, indicated by its prompt (**telnet>**). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see below) with those arguments.

Once a connection has been opened, **telnet** enters input mode. In this mode, text typed is sent to the remote host. The input mode entered will be either "character at a time" or "line by line" depending on what the remote system supports.

In "character at a time" mode, most text typed is immediately sent to the remote host for processing.

In "line by line" mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The "local echo character" (initially '^E') may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the *localchars* toggle is TRUE (the default in line mode; see below), the user's **quit**, **intr**, and **flush** characters are trapped locally, and sent as TELNET protocol sequences to the remote side. There are options (see **toggle autoflush** and **toggle autosynch** below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of **quit** and **intr**).

While connected to a remote host, **telnet** command mode may be entered by typing the **telnet** "escape character" (initially '^]', (control-right-bracket)). When in command mode, the normal terminal editing conventions are available.

## USAGE

### Telnet Commands

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the **mode**, **set**, **toggle**, and **display** commands).

**open** *host* [ *port* ]

Open a connection to the named host. If no port number is specified, **telnet** will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see **hosts**(5)) or an Internet address specified in the "dot notation" (see **inet**(3N)).

**close** Close a TELNET session and return to command mode.

**quit** Close any open TELNET session and exit **telnet**. An EOF (in command mode) will also close a session and exit.

**z** Suspend **telnet**. This command only works when the user is using a shell that supports job control, such as **csh**(1).

**mode** *type*

*type* is either **line** (for "line by line" mode) or *character* (for "character at a time" mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

**status** Show the current status of **telnet**. This includes the peer one is connected to, as well as the current mode.

**display** [ *argument...* ]
Display all, or some, of the **set** and **toggle** values (see below).

**?** [ *command* ]
Get help. With no arguments, **telnet** prints a help summary. If a command is specified, **telnet** will print the help information for just that command.

**send** *arguments*
Send one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

*escape* Send the current **telnet** escape character (initially "^]").

*synch* Send the TELNET **SYNCH** sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system -- if it does not work, a lower case "r" may be echoed on the terminal).

**brk** Send the TELNET **BRK** (Break) sequence, which may have significance to the remote system.

**ip** Send the TELNET **IP** (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

*ao* Sends the TELNET **AO** (Abort Output) sequence, which should cause the remote system to flush all output **from** the remote system **to** the user's terminal.

*ayt* Sends the TELNET **AYT** (Are You There) sequence, to which the remote system may or may not choose to respond.

**ec** Sends the TELNET **EC** (Erase Character) sequence, which should cause the remote system to erase the last character entered.

*el* Sends the TELNET **EL** (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

*ga* Sends the TELNET **GA** (Go Ahead) sequence, which likely has no significance to the remote system.

*nop* Sends the TELNET **NOP** (No Operation) sequence.

*?* Prints out help information for the **send** command.

**set** *argument value*
Set any one of a number of **telnet** variables to a specific value. The special value "off" turns off the function associated with the variable. The values of variables may be interrogated with the **display** command. The variables which may be specified are:

**echo** This is the value (initially "^E") which, when in "line by line" mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

*escape* This is the **telnet** escape character (initially "^[") which causes entry into **telnet** command mode (when connected to a remote system).

*interrupt*
If **telnet** is in *localchars* mode (see **toggle** *localchars* below) and the *interrupt* character is typed, a TELNET **IP** sequence (see **send ip** above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

*quit* If **telnet** is in *localchars* mode (see **toggle** *localchars* below) and the *quit* character is typed, a TELNET **BRK** sequence (see **send brk** above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's **quit** character.

*flushoutput*
If **telnet** is in *localchars* mode (see **toggle** *localchars* below) and the *flushoutput* character is typed, a TELNET **AO** sequence (see **send** *ao* above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's **flush** character.

*erase* If **telnet** is in *localchars* mode (see **toggle** *localchars* below), **and if telnet** is operating in "character at a time" mode, then when this character is typed, a TELNET **EC** sequence (see **send ec** above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's **erase** character.

**kill** If **telnet** is in *localchars* mode (see **toggle** *localchars* below), **and if telnet** is operating in "character at a time" mode, then when this character is typed, a TELNET **EL** sequence (see **send** *el* above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's **kill** character.

*eof* If **telnet** is operating in "line by line" mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's **eof** character.

**toggle** *arguments...*
Toggle (between TRUE and FALSE) various flags that control how **telnet** responds to events. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

*localchars*
If this is TRUE, then the **flush**, **interrupt**, **quit**, **erase**, and **kill** characters (see **set** above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively *ao*, *ip*, *brk*, *ec*, and *el*; see **send** above). The initial value for this toggle is TRUE in "line by line" mode, and FALSE in "character at a time" mode.

*autoflush*
If *autoflush* and *localchars* are both TRUE, then when the *ao*, *intr*, or *quit* characters are recognized (and transformed into TELNET sequences; see **set** above for details), **telnet** refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET *Timing Mark* option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an "stty noflsh", otherwise FALSE (see **stty(1V)**).

*autosynch*
If *autosynch* and *localchars* are both TRUE, then when either the *intr* or *quit* characters is typed (see **set** above for descriptions of the *intr* and *quit* characters), the resulting TELNET sequence sent is followed by the TELNET **SYNCH** sequence. This procedure **should** cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

*crmod* Toggle RETURN mode. When this mode is enabled, most RETURN characters received from the remote host will be mapped into a RETURN followed by a LINEFEED. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends RETURN, but never LINEFEED. The initial value for this toggle is FALSE.

*debug* Toggle socket level debugging (useful only to the super-user). The initial value for this toggle is FALSE.

*options* Toggle the display of some internal **telnet** protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.

*netdata* Toggle the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

*?* Display the legal **toggle** commands.

**SEE ALSO**

**csh**(1), **rlogin**(1C), **stty**(1V) **inet**(3N), **hosts**(5)

**BUGS**

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in "line by line" mode.

There is enough settable state to justify a **.telnetrc** file.

No capability for a **.telnetrc** file is provided.

In "line by line" mode, the terminal's EOF character is only recognized (and sent to the remote system) when it is the first character on a line.

## NAME

test – return true or false according to a conditional expression

## SYNOPSIS

**test** *expression*

[ *expression* ]

## DESCRIPTION

**test** evaluates the expression *expression* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned. **test** returns a non-zero exit if there are no arguments.

## USAGE

### Primitives

The following primitives are used to construct *expression*.

**–b** *filename*
True if *filename* exists and is a block special device.

**–c** *filename*
True if *filename* exists and is a character special device.

**–d** *filename*
True if *filename* exists and is a directory.

**–f** *filename*
True if *filename* exists and is not a directory.

**–g** *filename*
True if *filename* exists and its set-group-ID bit is set.

**–h** *filename*
True if *filename* exists and is a symbolic link.

**–k** *filename*
True if *filename* exists and its sticky bit is set.

**–l** *string* the length of the string.

**–n** *s1* True if the length of the string *s1* is non-zero.

**–p** *filename*
True if *filename* exists and is a named pipe (FIFO).

**–r** *filename*
True if *filename* exists and is readable.

**–s** *filename*
True if *filename* exists and has a size greater than zero.

**–t** [ *fildes* ]
True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

**–u** *filename*
True if *filename* exists and its set-user-ID bit is set.

**–w** *filename*
True if *filename* exists and is writable.

**–x** *filename*
True if *filename* exists and is executable.

**−z** *s1* True if the length of string *s1* is zero.

*s1* **=** *s2* True if the strings *s1* and *s2* are equal.

*s1* **!=** *s2* True if the strings *s1* and *s2* are not equal.

*s1* True if *s1* is not the null string.

*n1* **−eq** *n2* True if the integers *n1* and *n2* are numerically equal.

*n1* **−ne** *n2* True if the integer *n1* is not numerically equal to the integer *n2*.

*n1* **−gt** *n2* True if the integer *n1* is numerically greater than the integer *n2*.

*n1* **−ge** *n2* True if the integer *n1* is numerically greater than or equal to the integer *n2*.

*n1* **−lt** *n2* True if the integer *n1* is numerically less than the integer *n2*.

*n1* **−le** *n2* True if the integer *n1* is numerically less than or equal to the integer *n2*.

**Operators**

The above primaries may be combined with the following operators:

**!** Unary negation operator.

**−a** Binary *and* operator.

**−o** Binary *or* operator.

(*expression*)
Parentheses for grouping.

**−a** has higher precedence than **−o**. Notice that all the operators and flags are separate arguments to **test**. Notice also that parentheses are meaningful to the Shell and must be escaped.

## SYSTEM V USAGE

The actions of the System V version of **test** are the same, except for the following primitives:

**−f** *filename* True if *filename* exists and is a regular file.

**−l** *string* Not supported.

## SEE ALSO

**find**(1), **sh**(1)

## WARNING

In the second form of the command (that is, the one that uses [ ], rather than the word **test**), the square brackets must be delimited by blanks.

Some UNIX systems do not recognize the second form of the command.

## NOTES

The **test** command is built into the Bourne shell, which chooses the 4.2 BSD or the System V version of **test**, depending on whether **/usr/5bin** appears before **/usr/bin** in the shell's **PATH** variable. This is consistent with the behavior of other commands present in both **/usr/bin** and **/usr/5bin**.

The fact that **test** is built into the shell also means that a program named **test** cannot be run without specifying a pathname; if the program is in the current directory, **./test** will suffice.

## NAME

textedit – SunView window- and mouse-based text editor

## SYNOPSIS

**textedit** [ *generic-tool-arguments* ] [ **–Ea on | off** ] [ **–adjust_is_pending_delete** ] [ **–Ei on | off** ]
[ **–auto_indent** ] [ **–Eo on | off** ] [ **–okay_to_overwrite** ] [ **–Er on | off** ] [ **–read_only** ]
[ **–Ec** *N* ] [ **–checkpoint** *count* ] [ **–EL** *lines* ] [ **–lower_context** *lines* ] [ **–Em** *pixels* ]
[ **–margin** *pixels* ] [ **–En** *N* ] [ **–number_of_lines** *lines* ] [ **–Es** *N* ] [ **–scratch_window** *lines* ]
[ **–ES** *N* ] [ **–multi_click_space** *radius* ] [ **–Et** *N* ] [ **–tab_width** *tabstop* ] [ **–ET** *N* ]
[ **–multi_click_timeout** *intrvl* ] [ **–Eu** *N* ] [ **–history_limit** *max* ] [ **–EU** *N* ]
[ **–upper_context** *lines* ] *filename*

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**textedit** is a mouse-oriented text editor that runs within the SunView environment. It creates a window containing two text subwindows. The top subwindow (referred to as the "scratch" window) can be used to store small pieces of text. The bottom subwindow (referred to as the "edit" window) displays the contents of *filename*, if given.

The name of the file currently being edited is displayed in the left-hand portion of the frame header. The name of the current working directory is displayed in the right-hand portion.

## OPTIONS

*generic-tool-arguments*
**textedit** accepts the SunView generic tool arguments listed in **sunview**(1).

**–Ea on | off**
**–adjust_is_pending_delete**
Choose whether or not an adjustment to a selection makes the selection "pending-delete." The default is off. This option corresponds to, and overrides, the **adjust_is_pending_delete** Text defaults entry.

**–Ei on | off**
**–auto_indent** Choose whether or not to automatically indent newly-opened lines. The default is off. Corresponds to the **auto_indent** Text default.

**–Eo on | off**
**–okay_to_overwrite**
Set behavior to the **Store as New File** menu item. If **on** a **Store as New File** to the current file is treated as a **Save Current File**. If **off** (the standard default), **Store as New File** operations using the current filename result in an error message. Corresponds to **Store_self_is_save**.

**–Er on | off**
**–read_only** Turn read-only mode on or off. When on, text cannot be modified.

**–Ec** *N*
**–checkpoint** *count*
Checkpoint after every *count* editing operations. If *count* is 0 (the standard default), no checkpointing takes place. Each character typed, each **Paste**, and each **Cut** counts as an editing operation. Corresponds to **checkpoint_frequency**.

**-EL** *lines*
**-lower_context** *lines*
Specify the minimum number of lines to keep between the caret and the bottom of the text subwindow. The default is 2. Corresponds to **lower_context**.

**-Em** *pixels*
**-margin** *pixels*
Set the scrollbar margin width in pixels. The default is 4. Corresponds to **left_margin**.

**-En** *N*
**-number_of_lines** *lines*
Set the number of lines in the bottom subwindow. The default is 45.

**-Es** *N*
**-scratch_window** *lines*
Set the number of lines in the scratch window. A zero value means that there is no scratch window. The standard default is 1. Corresponds to **scratch_window**.

**-ES** *N*
**-multi_click_space** *radius*
Set the radius, in pixels, within which clicks must occur to be treated as a multi-click selection. The default is 3 pixels. Corresponds to **multi_click_space**.

**-Et** *N*
**-tab_width** *tabstop*
Set the number of SPACE characters displayed per TAB stop. The default is 8. This option has no effect on the characters in the file. Corresponds to **tab_width**.

**-ET** *N*
**-multi_click_timeout** *intrvl*
Set the interval, in milliseconds, within which any two clicks must occur to be treated as a multi-click selection. The default is 390 milliseconds. Corresponds to **multi_click_timeout**.

**-Eu** *N*
**-history_limit** *max*
Set the maximum number of editing operations that can be undone or replayed. The default is 50. Corresponds to **history_limit**.

**-EU N**
**-upper_context** *lines*
Set the minimum number of lines to keep between the caret and the top of the text subwindow. The default is 2. Corresponds to **upper_context**.

## USAGE

For a description of how to use the facilities of the text subwindows, see the *SunView User's Guide*.

### Signal Processing

If **textedit** hangs, for whatever reason, you can send a **SIGHUP** signal to its process ID, which forces it to write any changes (if possible):

**kill -HUP** *pid*

The edits are written to the file **textedit.***pid* in its working directory. If that fails, **textedit** successively tries to write to a file by that name in **/var/tmp**, and then **/tmp**. In addition, whenever **textedit** catches a fatal signal, such as SIGILL, it tries to write out the edits before aborting.

**Defaults Options**

There are several dozen user-specified defaults that affect the behavior of the text-based facilities. See **defaultsedit**(1) for a complete description. Important defaults entries in the **Text** category are:

**Edit_back_char** Set the character for erasing to the left of the caret. The standard default is DELETE. Note: the tty erase character-setting has no effect on **textedit**. Text-based tools refer only to the defaults database key settings.

**Edit_back_word** Set the character for erasing the word to the left of the caret. The standard default is CTRL-W.

**Edit_back_line** Set the character for erasing all characters to the left of the caret. The standard default is CTRL-U.

**Checkpoint_frequency**

If set to **0** (the standard default) no checkpointing is done. For any value greater than zero, a checkpoint is made each time the indicated number of editing operations has been performed since the last checkpoint. Each character typed, each **Paste**, and each **Cut** counts as an editing operation. The checkpoint file has a name of the form: *filename*%%, where *filename* is the name of the file being edited.

**Making a selection**

In **textedit**, the mouse is used to specify a selection, which is a character span to operate on. The mouse is also used to position the insertion point and to invoke a menu of commands.

The assignment of commands to the mouse buttons is:

| Mouse button | Description |
|---|---|
| LEFT | Starts a new selection and moves the insertion point to the end of the selection nearest the mouse cursor. |
| MIDDLE | Extends a selection, and moves the insertion point. |
| RIGHT | Displays a menu of operations, explained below. |

There are two types of selections: a primary selection is indicated by video-inversion of the span of characters, and tends to persist. A secondary selection is indicated by underlining the span of characters and only exists while one of the four function keys corresponding to the commands **Cut**, **Find**, **Paste**, or **Copy**, is depressed.

In addition, a selection can be "pending-delete," as indicated by overlaying the span of characters with a light gray pattern. A selection is made pending-delete by holding the CTRL key while clicking the LEFT or MIDDLE mouse buttons. If a primary selection is pending-delete, it is only deleted when characters are inserted, either by type-in or by **Paste** or **Copy**. If a secondary selection is pending-delete, it is deleted when the function key is released, except in the case of the **Find**, which deselects the secondary selection.

You can make adjusted selections switch to pending-delete using the **adjust_is_pending_delete** defaults entry, or the **–Ea** option. In this case, CTRL-Middle makes the selection *not* pending-delete.

Commands that operate on the primary selection do so even if the primary selection is not in the window that issued the command.

**Inserting Text and Command Characters**

For the most part, typing any of the standard keys either inserts the corresponding character at the insertion point, or erases characters. However, certain key combinations are treated as commands. Some of the most useful are:

| Command | Character | Description |
|---|---|---|
| **Cut-Primary** | META-X | Erases, and moves to the Clipboard, the primary selection. |
| **Find-Primary** | META-F | Searches the text for the pattern specified by the primary selection or by the Clipboard, if there is no primary selection. |
| **Copy-to-Clipboard** | META-C | Copies the primary selection to the Clipboard. |
| **Paste-Clipboard** | META-V | Inserts the Clipboard contents at the insertion point. |
| **Copy-then-Paste** | META-P | Copies the primary selection to the insertion point (through the Clipboard). |
| **Go-to-EOF** | CTRL-RETURN | Moves the insertion point to the end of the text, positioning the text so that the insertion point is visible. |

**Function Keys**

The commands indicated by use of the function keys are:

| Command | Sun-2\|3 Key | Description |
|---|---|---|
| **Stop** | L1 | Aborts the current command. |
| **Again** | L2 | Repeats the previous editing sequence since a primary selection was made. |
| **Undo** | L4 | Undoes a prior editing sequence. |
| **Front** | L5 | Makes the window completely visible (or hides it, if it is already exposed). |
| **Copy** | L6 | Copies the primary selection, either to the Clipboard or at the closest end of the secondary selection. |
| **Open** | L7 | Makes the window iconic (or normal, if it is already iconic). |
| **Paste** | L8 | Copies either the secondary selection or the Clipboard at the insertion point. |
| **Find** | L9 | Searches for the pattern specified by, in order, the secondary selection, the primary selection, or the Clipboard. |
| **Cut** | L10 | Erases, and moves to the Clipboard, either the primary or the secondary selection. |
| CAPSLOCK | F1 | Forces all subsequently typed alphabetic characters to be upper-case. This key is a toggle; striking it a second time undoes the effect of the first strike. |

**Find** usually searches the text forwards, towards the end. Holding down the SHIFT key while invoking **Find** searches backward through the text, towards the beginning. If the pattern is not found before the search encounters either extreme, it "wraps around" and continues from the other extreme. **Find** starts the search at the appropriate end of the primary selection, if the primary selection is in the subwindow that the search is made in; otherwise it starts at the insertion point, unless the subwindow cannot be edited, in which case it starts at the beginning of the text.

CTRL-Find invokes the **Find and Replace** pop-up frame.

The default assignment of function keys can be modified using **defaultsedit**(1).

**Menu Items**

**File** A pull-right menu item for file operations.

**Edit** A pull-right menu item equivalent of the editing function keys. The **Edit** submenu provides **Again, Undo, Copy, Paste,** and **Cut** (same as function keys L2, L4, L6, L8, and L10).

**Display** A pull-right menu item for controlling the way text is displayed and line display format.

**Find** A pull-right menu item for find and delimiter matching operations.

**Extras** A user definable pull-right menu item. The **Extras** standard submenu is controlled by **/usr/lib/text_extras_menu**. This file has the same syntax as **.rootmenu** file. See **sunview**(1).

Only those items that are active appear as normal text in the menu; inactive items (which are inappropriate at the time) are "grayed out".

**User Defined Commands**

The file **/usr/lib/text_extras_menu** specifies filter programs that are included in the text subwindow **Extras** pull-right menu item. The file **~/.textswrc** specifies filter programs that are assigned to (available) function keys. These filters are applied to the contents of the primary selection. Their output is entered at the caret.

The file **/usr/lib/textswrc** is a sample containing a set of useful filters. It is not read automatically.

## ENVIRONMENT

The environment variables **LC_CTYPE, LANG,** and **LC_default** control the character classification throughout **textedit**. On entry to **textedit**, these environment variables are checked in the following order: **LC_CTYPE, LANG,** and **LC_default.** When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## FILES

| | |
|---|---|
| **~/.textswrc** | specifies bindings of filters to function keys |
| **/usr/lib/text_extras_menu** | specifies bindings of filters for the extras menu pull-right items |
| **/usr/bin** | contains useful filters, including **shift_lines** and **capitalize.** |
| *filename%* | prior version of *filename* is available here after a **Save Current File** menu operation |
| **textedit.***pid* | edited version of *filename*; generated in response to fatal internal errors |
| **/tmp/Text*** | editing session logs |

## SEE ALSO

**defaultsedit**(1), **kill**(1), **sunview**(1)

*SunView User's Guide*

## DIAGNOSTICS

**Cannot open file '***filename***', aborting!** *filename* does not exist or cannot be read.

**textedit** produces the following exit status codes:

| | |
|---|---|
| 0 | normal termination |
| 1 | standard SunView help message was printed |
| 2 | help message was requested and printed |
| 3 | abnormal termination in response to a signal, usually due to an internal error |
| 4 | abnormal termination during initialization, usually due to a missing file or running out of swap space |

## BUGS

Multi-click to change the current selection does not work for **Adjust Selection.**

Handling of long lines is incorrect in certain scrolling situations.

There is no way to replay any editing sequence except the most recent.

'**textedit** *newfile*' fails if *newfile* does not exist.

## NAME

textedit_filters, align_equals, capitalize, insert_brackets, remove_brackets, shift_lines – filters provided with textedit(1)

## SYNOPSIS

**align_equals**

**capitalize** [ **–u** | **–l** | **–c** ]

**insert_brackets** *l r*

**remove_brackets** *l r*

**shift_lines** [ **–t** ] *n*

## DESCRIPTION

Each of these filters is designed to operate on pending-delete selections in text subwindows. You can use them from within text subwindows either by mapping them to function keys in your **.textswrc** file or adding them to the text 'Extras' menu in your **.text_extras_menu** file. See the *SunView User's Guide* for details. When a filter is used as a command (perhaps in a pipeline), it is applied to the standard input and the filtered text appears on standard output.

**align_equals** lines up the '=' (equal signs) in C assignment statements. Some programmers feel that this makes for improved readability. It aligns all equal signs with the rightmost equal sign in the selection (or the standard input), by padding with spaces between the sign and the previous nonwhite character; it replaces the selection with the aligned text (or writes this text to the standard output). For instance:

```
big_long_expression = x;
shorter_expr = y;
z += 1;
```

becomes:

```
big_long_expression = x;
shorter_expr        = y;
z                  += 1;
```

**capitalize** changes the capitalization of the selection (or the standard input) and replaces it (or writes to the standard output). The **–l** option converts all characters to lower case; **–c** converts the first letter of each word to upper case; and **–u** converts all characters to upper case. If no option is specified, then **capitalize** consults its input to determine what to do. If the text is all capitals, it is converted to all lower case. If the text is all lower case or of mixed cases and contains no white space (such as a NEWLINE, SPACE, or TAB), it is converted to all capitals. If there is white space, then the case of the first character in each word is inverted.

**insert_brackets** surrounds the selection (or the standard input) with the specified character sequences. *l* and *r* are the left- and right-bracketing characters, respectively.

**remove_brackets** removes the left- and right-bracketing characters, specified by *l* and *r*, respectively from the selection (or the standard input).

**shift_lines** adjusts indentation of the selection (or the standard input) by *n* spaces, and replaces the selection with the adjusted text (or writes to the standard output). **shift_lines** adjusts to the left when *n* is negative. If **–t** is specified, the lines are shifted left or right by *n* tab stops. The default is 8 spaces per tab stop, but if the first line of the selection (or the standard input) begins with white space, then the tab stops are set to four spaces.

**FILES**

| | |
|---|---|
| **/tmp/Cap.*pid*** | temporary file used by **capitalize** |
| **/tmp/Ins.*pid*** | temporary file used by **insert_brackets** |
| **/usr/lib/.text_extras_menu** | default 'Extras' menu |
| **/usr/lib/.textswrc** | sample function-key mappings |

**SEE ALSO**

**textedit(1)**

*SunView User's Guide*

## NAME

tftp – trivial file transfer program

## SYNOPSIS

**tftp** [ *host* ]

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**tftp** is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case **tftp** uses *host* as the default host for future transfers (see the **connect** command below).

## USAGE

### Commands

Once **tftp** is running, it issues the prompt: **tftp>** and recognizes the following commands:

**connect** *host-name* [ *port* ]

Set the *host* (and optionally *port*) for transfers. Note: the TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the **connect** command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the **connect** command; the remote host can be specified as part of the **get** or **put** commands.

**mode** *transfer-mode*

Set the mode for transfers; *transfer-mode* may be one of **ascii** or **binary**. The default is **ascii**.

**put** *filename*
**put** *localfile remotefile*
**put** *filename1 filename2 ... filenameN remote-directory*

Transfer a file, or a set of files, to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host if the host has already been specified, or a string of the form

*host*:*filename*

to specify both a host and filename at the same time. If the latter form is used, the specified host becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be running the UNIX system.

**get** *filename*
**get** *remotename localname*
**get** *filename1 filename2 ... filenameN*

Get a file or set of files from the specified remote *sources*. *source* can be in one of two forms: a filename on the remote host if the host has already been specified, or a string of the form

*host*:*filename*

to specify both a host and filename at the same time. If the latter form is used, the last host specified becomes the default for future transfers.

**quit** Exit **tftp**. An EOF also exits.

**verbose** Toggle verbose mode.

**trace** Toggle packet tracing.

**status** Show current status.

**rexmt** *retransmission-timeout*
Set the per-packet retransmission timeout, in seconds.

**timeout** *total-transmission-timeout*
Set the total transmission timeout, in seconds.

**ascii** Shorthand for **mode ascii**.

**binary** Shorthand for **mode binary**.

**?** [ *command-name* ... ]
Print help information.

WARNING

The default *transfer-mode* is **ascii**. This differs from pre-4.0 Sun (and pre-4.3 BSD) releases, so explicit action must now be taken when transferring non-ASCII files such as executable commands.

BUGS

Because there is no user-login or validation within the TFTP protocol, many remote sites restrict file access in various ways. Approved methods for file access are specific to each site, and therefore cannot be documented here.

## NAME

time – time a command

## SYNOPSIS

**time** [ *command* ]

## SYSTEM V SYNOPSIS

**/usr/5bin/time** [ *command* ]

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

There are three distinct versions of **time**: it is built in to the C shell, and is an executable program available in **/usr/bin/time** and **/usr/5bin/time** when using the Bourne shell. In each case, times are displayed on the diagnostic output stream.

In the case of the C shell, a **time** command with no *command* argument simply displays a summary of time used by this shell and its children. When arguments are given the specified simple *command* is timed and the C shell displays a time summary as described in **csh**(1).

The **time** commands in **/usr/bin/time** and **/usr/5bin/time** time the given *command*, which must be specified, that is, *command* is not optional as it is in the C shell's timing facility. When the command is complete, **time** displays the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds. The only difference between the versions in **/usr/bin/time** and **/usr/5bin/time** is between their output formats; **/usr/bin/time** prints all three times on the same line, while **/usr/5bin/time** prints them on separate lines.

## EXAMPLES

The three examples here show the differences between the **csh** version of **time** and the versions in **/usr/bin/time** and **/usr/bin/time**. The example assumes that **csh** is the shell in use.

```
example% time wc /usr/share/man/man1/csh.1
1876  11223  65895 /usr/share/man/man1/csh.1
2.7u 0.9s 0:03 91% 3+5k 19+2io 1pf+0w
example%/usr/bin/time wc /usr/share/man/man1/csh.1
1876  11223  65895 /usr/share/man/man1/csh.1
4.3 real       2.7 user       1.0 sys
example% /usr/5bin/time wc /usr/share/man/man1/csh.1
1876  11223  65895 /usr/share/man/man1/csh.1
real      4.3
user      2.7
sys       1.0
example%
```

## SEE ALSO

**csh**(1)

**BUGS**

Elapsed time is accurate to the second, while the CPU times are measured to the 50th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

When the command being timed is interrupted, the timing values displayed may not always be accurate.

## NAME

tip – connect to remote system

## SYNOPSIS

**tip** [ –v ] [ *–speed-entry* ] *hostname* | *phone-number*

## DESCRIPTION

**tip** establishes a full-duplex terminal connection to a remote host. Once the connection is established, a remote session using **tip** behaves like an interactive session on a local terminal.

The *remote* file (described in the **remote**(5) manual page) contains entries describing remote systems and line speeds used by **tip**.

Each host has a default baud rate for the connection, or you can specify a speed with the *–speed-entry* command line argument.

When *phone-number* is specified, **tip** looks for an entry in the *remote* file of the form:

**tip** *–speed-entry*

When it finds such an entry, it sets the connection speed accordingly. If it finds no such entry, **tip** interprets *–speed-entry* as if it were a system name, resulting in an error message.

If you omit *–speed-entry*, **tip** uses the **tip0** entry to set a speed for the connection.

When establishing the connection **tip** sends a connection message to the remote system. The default value for this message can be found in the *remote* file.

When **tip** attempts to connect to a remote system, it opens the associated device with an exclusive-open **ioctl**(2) call. Thus only one user at a time may access a device. This is to prevent multiple processes from sampling the terminal line. In addition, **tip** honors the locking protocol used by **uucp**(1C).

When **tip** starts up it reads commands from the file **.tiprc** in your home directory.

## OPTIONS

**–v** Display commands from the **.tiprc** file as they are executed.

## USAGE

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well).

At any time that **tip** prompts for an argument (for example, during setup of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, aborts the dialogue and returns you to the remote machine.

### Commands

A tilde (**~**) appearing as the first character of a line is an escape signal which directs **tip** to perform some special action. **tip** recognizes the following escape sequences:

**~^D**

**~.** Drop the connection and exit (you may still be logged in on the remote machine).

**~c** [*name*]

Change directory to *name* (no argument implies change to your home directory).

**~!** Escape to an interactive shell on the local machine (exiting the shell returns you to **tip**).

**~>** Copy file from local to remote.

**~<** Copy file from remote to local.

~**p** *from* [ *to* ]
Send a file to a remote host running the UNIX system. When you use the put command, the remote system runs the command string

**cat** > *to*

while **tip** sends it the *from* file. If the *to* file is not specified, the *from* file name is used. This command is actually a UNIX-system-specific version of the '~>' command.

~**t** *from* [ *to* ]
Take a file from a remote host running the UNIX system. As in the put command the *to* file defaults to the *from* file name if it is not specified. The remote host executes the command string

**cat** *from* **; echo** ^**A**

to send the file to **tip**.

~| Pipe the output from a remote command to a local process. The command string sent to the local system is processed by the shell.

~**C** Connect a program to the remote machine. The command string sent to the program is processed by the shell. The program inherits file descriptors 0 as remote line input, 1 as remote line output, and 2 as tty standard error.

~**$** Pipe the output from a local process to the remote host. The command string sent to the local system is processed by the shell.

~**#** Send a BREAK to the remote system.

~**s** Set a variable (see the discussion below).

~^**Z** Stop **tip** (only available when run under a shell that supports job control, such as the C shell).

~^**Y** Stop only the "local side" of **tip** (only available when run under a shell that supports job control, such as the C shell); the "remote side" of **tip**, the side that displays output from the remote host, is left running.

~**?** Get a summary of the tilde escapes.

Copying files requires some cooperation on the part of the remote host. When a ~> or ~< escape is used to send a file, **tip** prompts for a file name (to be transmitted or received) and a command to be sent to the remote system, in case the file is being transferred from the remote system. The default end of transmission string for transferring a file from the local system to the remote is specified as the **oe** capability in the **remote**(5) file, but may be changed by the set command. While **tip** is transferring a file the number of lines transferred will be continuously displayed on the screen. A file transfer may be aborted with an interrupt.

AUTO-CALL UNITS

**tip** may be used to dial up remote systems using a number of auto-call unit's (ACU's). When the remote system description contains the **du** capability, **tip** uses the call-unit (**cu**), ACU type (**at**), and phone numbers (**pn**) supplied. Normally **tip** displays verbose messages as it dials. See **remote**(5) for details of the remote host specification.

Depending on the type of auto-dialer being used to establish a connection the remote host may have garbage characters sent to it upon connection. The user should never assume that the first characters typed to the foreign host are the first ones presented to it. The recommended practice is to immediately type a **kill** character upon establishing a connection (most UNIX systems either support @ or CTRL-U as the initial kill character).

**tip** currently supports the Ventel MD-212+ modem and DC Hayes-compatible modems.

## REMOTE HOST DESCRIPTIONS

Descriptions of remote hosts are normally located in the system-wide file **/etc/remote**. However, a user may maintain personal description files (and phone numbers) by defining and exporting the REMOTE shell variable. The *remote* file must be readable by **tip**, but a secondary file describing phone numbers may be maintained readable only by the user. This secondary phone number file is **/etc/phones**, unless the shell variable PHONES is defined and exported. As described in **remote**(5), the *phones* file is read when the host description's phone number(s) capability is an '**@**'. The phone number file contains lines of the form:

*system-name phone-number*

Each phone number found for a system is tried until either a connection is established, or an end of file is reached. Phone numbers are constructed from '**0123456789-=***', where the '**=**' and '*****' are used to indicate a second dial tone should be waited for (ACU dependent).

## TIP INTERNAL VARIABLES

**tip** maintains a set of variables which are used in normal operation. Some of these variables are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the ~s escape. The syntax for variables is patterned after **vi**(1) and **mail**(1). Supplying **all** as an argument to the ~s escape displays all variables that the user can read. Alternatively, the user may request display of a particular variable by attaching a **?** to the end. For example '**~s escape?**' displays the current escape character.

Variables are numeric, string, character, or Boolean values. Boolean variables are set merely by specifying their name. They may be reset by prepending a **!** to the name. Other variable types are set by appending an **=** and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables.

Variables may be initialized at run time by placing set commands (without the ~s prefix) in a **.tiprc** file in one's home directory. The **-v** option makes **tip** display the sets as they are made. Comments preceded by a **#** sign can appear in the **.tiprc** file.

Finally, the variable names must either be completely specified or an abbreviation may be given. The following list details those variables known to **tip**.

**beautify**

(bool) Discard unprintable characters when a session is being scripted; abbreviated **be**. If the **nb** capability is present, **beautify** is initially set to *off*; otherwise, **beautify** is initially set to *on*.

**baudrate**

(num) The baud rate at which the connection was established; abbreviated **ba**. If a baud rate was specified on the command line, **baudrate** is initially set to the specified value; otherwise, if the **br** capability is present, **baudrate** is initially set to the value of that capability; otherwise, **baudrate** is set to 300 baud. Once **tip** has been started, **baudrate** can only changed by the super-user.

**dialtimeout**

(num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated **dial**. **dialtimeout** is initially set to 60 seconds, and can only changed by the super-user.

**disconnect**

(str) The string to send to the remote host to disconnect from it; abbreviated **di**. If the **di** capability is present, **disconnect** is initially set to the value of that capability; otherwise, **disconnect** is set to a null string ("").

**echocheck**

(bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; abbreviated **ec**. If the **ec** capability is present, **echocheck** is initially set to *on*; otherwise, **echocheck** is initially set to *off*.

**eofread** (str) The set of characters which signify an end-of-transmission during a ~< file transfer command; abbreviated **eofr**. If the **ie** capability is present, **eofread** is initially set to the value of that capability; otherwise, **eofread** is set to a null string ("").

**eofwrite**
(str) The string sent to indicate end-of-transmission during a ~> file transfer command; abbreviated **eofw**. If the **oe** capability is present, **eofread** is initially set to the value of that capability; otherwise, **eofread** is set to a null string ("").

**eol** (str) The set of characters which indicate an end-of-line. **tip** will recognize escape characters only after an end-of-line. If the **el** capability is present, **eol** is initially set to the value of that capability; otherwise, **eol** is set to a null string ("").

**escape** (char) The command prefix (escape) character; abbreviated **es**. If the **es** capability is present, **escape** is initially set to the value of that capability; otherwise, **escape** is set to '~'.

**etimeout**
(num) The amount of time, in seconds, that **tip** should wait for the echo-check response when **echocheck** is set; abbreviated **et**. If the **et** capability is present, **etimeout** is initially set to the value of that capability; otherwise, **etimeout** is set to 10 seconds.

**exceptions**
(str) The set of characters which should not be discarded due to the beautification switch; abbreviated **ex**. If the **ex** capability is present, **exceptions** is initially set to the value of that capability; otherwise, **exceptions** is set to '\t\n\f\b'.

**force** (char) The character used to force literal data transmission; abbreviated **fo**. If the **fo** capability is present, **force** is initially set to the value of that capability; otherwise, **force** is set to \377 (which disables it).

**framesize**
(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated **fr**. If the **fs** capability is present, **framesize** is initially set to the value of that capability; otherwise, **framesize** is set to 1024.

**halfduplex**
(bool) Do local echoing because the host is half-duplex; abbreviated **hdx**. If the **hd** capability is present, **halfduplex** is initially set to *on*; otherwise, **halfduplex** is initially set to *off*.

**hardwareflow**
(bool) Do hardware flow control; abbreviated **hf**. If the **hf** capability is present, **hardwareflow** is initially set to *on*; otherwise, **hardwareflowcontrol** is initially set to *off*.

**host** (str) The name of the host to which you are connected; abbreviated **ho**. **host** is permanently set to the name given on the command line or in the HOST environment variable.

**localecho**
(bool) A synonym for **halfduplex**; abbreviated **le**.

**log** (str) The name of the file to which to log information about outgoing phone calls. **log** is initially set to **/var/adm/aculog**, and can only be inspected or changed by the super-user.

**parity** (str) The parity to be generated and checked when talking to the remote host; abbreviated **par**. The possible values are:

**none**
**zero** Parity is not checked on input, and the parity bit is set to zero on output.

**one** Parity is not checked on input, and the parity bit is set to one on output.

**even** Even parity is checked for on input and generated on output.

**odd** Odd parity is checked for on input and generated on output.

If the **pa** capability is present, **parity** is initially set to the value of that capability; otherwise, **parity** is set to **none**.

**phones** The file in which to find hidden phone numbers. If the environment variable PHONES is set, **phones** is set to the value of PHONES; otherwise, **phones** is set to **/etc/phones**. The value of **phones** cannot be changed from within **tip**.

**prompt** (char) The character which indicates an end-of-line on the remote host; abbreviated **pr**. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on receipt of this character. If the **pr** capability is present, **prompt** is initially set to the value of that capability; otherwise, **prompt** is set to **\n**.

**raise** (bool) Upper case mapping mode; abbreviated **ra**. When this mode is enabled, all lower case letters will be mapped to upper case by **tip** for transmission to the remote machine. If the **ra** capability is present, **raise** is initially set to *on*; otherwise, **raise** is initially set to *off*.

**raisechar**
(char) The input character used to toggle upper case mapping mode; abbreviated **rc**. If the **rc** capability is present, **raisechar** is initially set to the value of that capability; otherwise, **raisechar** is set to **\377** (which disables it).

**rawftp** (bool) Send all characters during file transfers; do not filter non-printable characters, and do not do translations like **\n** to **\r**. Abbreviated **raw**. If the **rw** capability is present, **rawftp** is initially set to *on*; otherwise, **rawftp** is initially set to *off*.

**record** (str) The name of the file in which a session script is recorded; abbreviated **rec**. If the **re** capability is present, **record** is initially set to the value of that capability; otherwise, **record** is set to **tip.record**.

**remote** The file in which to find descriptions of remote systems. If the environment variable REMOTE is set, **remote** is set to the value of REMOTE; otherwise, **remote** is set to **/etc/remote**. The value of **remote** cannot be changed from within **tip**.

**script** (bool) Session scripting mode; abbreviated **sc**. When **script** is *on*, **tip** will record everything transmitted by the remote machine in the script record file specified in **record**. If the **beautify** switch is on, only printable ASCII characters will be included in the script file (those characters between 040 and 0177). The variable **exceptions** is used to indicate characters which are an exception to the normal beautification rules. If the **sc** capability is present, **script** is initially set to *on*; otherwise, **script** is initially set to *off*.

**tabexpand**
(bool) Expand TAB characters to SPACE characters during file transfers; abbreviated **tab**. When **tabexpand** is *on*, each tab is expanded to 8 SPACE characters. If the **tb** capability is present, **tabexpand** is initially set to *on*; otherwise, **tabexpand** is initially set to *off*.

**tandem** (bool) Use XON/XOFF flow control to limit the rate that data is sent by the remote host; abbreviated **ta**. If the **nt** capability is present, **tandem** is initially set to *off*; otherwise, **tandem** is initially set to *on*.

**verbose** (bool) Verbose mode; abbreviated **verb**; When verbose mode is enabled, **tip** prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more. If the **nv** capability is present, **verbose** is initially set to *off*; otherwise, **verbose** is initially set to *on*.

**SHELL** (str) The name of the shell to use for the **~!** command; default value is **/bin/sh**, or taken from the environment.

**HOME** (str) The home directory to use for the **~c** command; default value is taken from the environment.

**EXAMPLES**

An example of the dialogue used to transfer files is given below.

**arpa% tip monet**
**[connected]**
*...(assume we are talking to a UNIX system)...*
**ucbmonet login: sam**
**Password:**
**monet% cat > sylvester.c**
**~> Filename: sylvester.c**
**32 lines transferred in 1 minute 3 seconds**
**monet%**
**monet% ~< Filename: reply.c**
**List command for remote host: cat reply.c**
**65 lines transferred in 2 minutes**
**monet%**
*...(or, equivalently)...*
**monet% ~p sylvester.c**
*...(actually echoes as ~[put] sylvester.c)...*
**32 lines transferred in 1 minute 3 seconds**
**monet%**
**monet% ~t reply.c**
*...(actually echoes as ~[take] reply.c)...*
**65 lines transferred in 2 minutes**
**monet%**
*...(to print a file locally)...*
**monet% ~|Local command: pr –h sylvester.c | lpr**
**List command for remote host: cat sylvester.c**
**monet% ~^D**
**[EOT]**
*...(back on the local system)...*

**ENVIRONMENT**

The following environment variables are read by **tip**.

**REMOTE** The location of the *remote* file.

**PHONES** The location of the file containing private phone numbers.

**HOST** A default host to connect to.

**HOME** One's log-in directory (for chdirs).

**SHELL** The shell to fork on a '~!' escape.

**FILES**

| | |
|---|---|
| **~/.tiprc** | initialization file |
| **/var/spool/locks/LCK ..*** | lock file to avoid conflicts with UUCP |
| **/var/adm/aculog** | file in which outgoing calls are logged |
| **/etc/phones** | |
| **/etc/remote** | |

**SEE ALSO**

**cu**(1C), **mail**(1), **uucp**(1C), **vi**(1), **ioctl**(2), **phones**(5), **remote**(5)

**BUGS**

There are two additional variables **chardelay** and **linedelay** that are currently not implemented.

## NAME

toolplaces – display current SunView window locations, sizes, and other attributes

## SYNOPSIS

**toolplaces** [ **–o|O|u** ] [ **–help** ]

## AVAILABILITY

This command is available with the *SunView User's Guide* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**toolplaces** generates position, size, label, and program attributes for the windows running on a SunView screen at the time of execution. (**toolplaces** does not work when SunView is not running.)

Many people redirect standard output from **toolplaces** to the **.sunview** file, so as to reuse the current window system attributes each time they execute **sunview**(1).

For each window on the screen at execution time, **toolplaces** shows:

the tool name
the "open" window position
the size of the window in pixels
the "closed," or icon, window position
an indicator of whether the window is open or closed
the label at the top of the window
the name of the program running in the window, if a program is running there
any flags or options to a program running in the window

**toolplaces** describes each window on one output line, as long as necessary, using the current **sunview** format.

Current **sunview** format consists of window tool descriptions, one per line, as in this example (the \ indicates that the current line continues on the next line):

```
shelltool –Wp 491 795 –Ws 580 87 –WP 0 836 –C
clock –Wp 120 120 –Ws 122 55 –WP 1086 826 –Wi \
        –Wl " open clock" –S –r –d wdm
shelltool –Wp 491 166 –Ws 650 567 –WP 702 836 \
        –Wl due rlogin due
shelltool –Wp 0 0 –Ws 650 525 –WP 64 836 \
        –Wl "Small Window: /usr/bin/csh"
shelltool –Wp 501 0 –Ws 650 812 –WP 128 836 \
        –Wl "Big Window: /usr/bin/csh"
```

## OPTIONS

**–o** Show window tool information in the old **suntools** format for window attributes, but specifies the appropriate tool names for each tool.

**–O** Show window tool information in the old **suntools** format for window attributes, specifying **toolname** as the name for each tool.

**–u** Show the updated window tool information in the order that you originally specified it.

**–help** Show help information preceding tool attributes.

**FILES**

| | |
|---|---|
| **~/.sunview** | format file for **sunview**(1) |

**SEE ALSO**

**sunview**(1)

*SunView User's Guide*

NAME

touch – update the access and modification times of a file

SYNOPSIS

**touch** [ **–c** ] [ **–f** ] *filename...*

SYSTEM V SYNOPSIS

**/usr/5bin/touch** [ **–acm** ] [ *date-time* ] *filename...*

AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

**touch** sets the access and modification times of each argument to the current time. A file is created if it does not already exist.

**touch** is valuable when used in conjunction with **make**(1), where, for instance, you might want to force a complete rebuild of a program composed of many pieces. In such a case, you might type:

```
example% touch *.c
example% make
```

**make**(1) would then see that all the **.c** files were more recent than the corresponding **.o** files, and would start the compilation from scratch.

OPTIONS

**–c** Do not create *filename* if it does not exist.

**–f** Attempt to force the touch in spite of read and write permissions on *filename*.

SYSTEM V OPTIONS

Options must be grouped to form the first argument. The *date-time* argument takes the form:

*mmddhhmm*[*yy*]

and updates the modification time to that specified, rather than the current time. The first *mm* is the month, **dd** is the day of the month, *hh* is the hour, and the second *mm* is the minute. If *yy* is specified, it is the last two digits of the year. If omitted, the current year is assumed.

**–a** Update only the access time.

**–c** Do not create *filename* if it does not exist.

**–m** Update only the modification time.

SEE ALSO

**make**(1), **utimes**(2)

BUGS

It is difficult to touch a file whose name consists entirely of digits in the System V **touch**, as it will interpret the first such non-flag argument as a time. You must ensure that there is a character in the name which is not a digit, by specifying it as *./name* rather than *name*.

## NAME

tput – initialize a terminal or query the terminfo database

## SYNOPSIS

**/usr/5bin/tput** [ **–T***type* ] *capability* [ *parameter* ... ]

**/usr/5bin/tput** [ **–T***type* ] **init**

**/usr/5bin/tput** [ **–T***type* ] **longname**

**/usr/5bin/tput** [ **–T***type* ] **reset**

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**tput** uses the **terminfo**(5V) database to make the values of terminal-dependent capabilities and information available to the shell, to initialize or reset the terminal, or return the long name of the requested terminal type. Normally, the terminal type is taken from the environment variable **TERM**; if the **–T***type* option is specified, the terminal type is taken from that option.

**tput** displays a string if the given *capability* is a string capability, or an integer if it is an integer capability; it displays no output if the capability is a boolean.

If *capability* is a boolean, **tput** returns true (0) if that capability is set, or false (1) otherwise. If *capability* is a string, **tput** returns true (0) if that capability is set, or false (1) otherwise. If *capability* is an integer, a value of true (0) is returned *whether or not* the capability is set for the terminal. To determine if an integer capability is set, you must examine the standard output.

For a complete list of capabilities and the *capability* associated with each, see **terminfo**(5V).

If *capability* is a string capability that takes parameters, the *parameter* arguments are instantiated into the string. An all-numeric *parameter* argument is passed to the attribute as a number.

## OPTIONS

**–T***type* Indicate the *type* of terminal. If this option is supplied, the environment variables **LINES** and **COLUMNS** are not used.

**init** If the **terminfo** database is present and an entry for the user's terminal exists, emit the terminal's initialization strings, set up any delays specified, and turn the expansion of TAB characters on or off, as specified by the terminal's entry in the **terminfo** database. If the terminal has a TAB character, and either has a capability for setting TAB characters or initially has its TAB characters set every 8 SPACE characters, expansion of TAB characters is turned off, otherwise expansion of TAB characters is turned on. If expansion of TAB characters is turned on, and the terminal has a capability for setting TAB characters, TAB stops are set to every eight columns. If an entry does not contain the information needed for any of these actions, that action is silently skipped.

**reset** Emit the terminal's reset strings, and set up delays and TAB characters as specified. If the reset strings are not present, but initialization strings are, the initialization strings are used.

**longname**

If the **terminfo** database is present and an entry for the user's terminal exists, emit the long name of the terminal. The long name is the last name in the first line of the terminal's description in the **terminfo** database.

**EXIT CODES**

| | |
|---|---|
| 0 | The boolean or string capability is set, or the capability is an integer type and is present. |
| 1 | The *capability* is not set. |
| 2 | Usage error. |
| 3 | The terminal is of an unknown type, or the **terminfo** database is not present. |
| 4 | Unknown **terminfo** capability. |
| −1 | The integer capability is not defined for this terminal type. |

**EXAMPLES**

| | |
|---|---|
| **tput init** | Initialize the terminal according to the type of terminal in the environmental variable **TERM**. This command can be included in a **.profile** or **.login** file. |
| **tput −Tsun reset** | Reset a Sun workstation console, **shelltool**(1) window, or **cmdtool**(1) window, overriding the type of terminal in the environmental variable **TERM**. |
| **tput cup 0 0** | Send the sequence to move the cursor to row **0**, column **0** (the upper left corner of the screen, usually known as the "home" cursor position). |
| **tput clear** | Echo the clear-screen sequence for the current terminal. |
| **tput cols** | Print the number of columns for the current terminal. |
| **tput −Tsun cols** | Print the number of columns for the Sun workstation console or subwindow. |
| **bold='tput smso'** | |
| **offbold='tput rmso'** | Set the shell variables **bold**, to begin stand-out mode sequence, and **offbold**, to end standout mode sequence, for the current terminal. This might be followed by a prompt: |

**echo "${bold}Please type in your name: ${offbold}\c"**

| | |
|---|---|
| **tput hc** | Set exit code to indicate if the current terminal is a hardcopy terminal. |
| **tput cup 23 4** | Send the sequence to move the cursor to row 23, column 4. |
| **tput longname** | Print the long name from the **terminfo** database for the type of terminal specified in the environmental variable **TERM**. |

**FILES**

| | |
|---|---|
| **/usr/share/lib/terminfo/?/*** | compiled terminal description database |
| **/usr/5include/curses.h** | **curses**(3V) header file |
| **/usr/5include/term.h** | **terminfo**(5V) header file |
| **/usr/share/lib/tabset/*** | TAB settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and TAB characters); for more information, see the **Tabs and Initialization** section of **terminfo**(5V). |
| **.login** | |
| **.profile** | |

**SEE ALSO**

**cmdtool**(1), **shelltool**(1), **stty**(1V), **curses**(3V), **terminfo**(5V)

## NAME

tr – translate characters

## SYNOPSIS

**tr** [ **–cds** ] [ *string1* [ *string2* ] ]

## SYSTEM V SYNOPSIS

**/usr/5bin/tr** [ **–cds** ] [ *string1* [ *string2* ] ]

## AVAILABILITY

The System V version of this command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**tr** copies the standard input to the standard output with substitution or deletion of selected characters. The arguments *string1* and *string2* are considered sets of characters. Any input character found in *string1* is mapped into the character in the corresponding position within *string2*. When *string2* is short, it is padded to the length of *string1* by duplicating its last character.

In either string the notation:

*a–b*

denotes a range of characters from *a* to *b* in increasing ASCII order. The character \, followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. As with the shell, the escape character \, followed by any other character, escapes any special meaning for that character.

## SYSTEM V DESCRIPTION

When *string2* is short, characters in *string1* with no corresponding character in *string2* are not translated.

In either string the following abbreviation conventions introduce ranges of characters or repeated characters into the strings. Note: in the System V version, square brackets are required to specify a range.

[*a–z*] Stands for the string of characters whose ASCII codes run from character *a* to character *z*, inclusive.

[*a***n*] Stands for *n* repetitions of *a*. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

## OPTIONS

Any combination of the options **–c**, **–d**, or **–s** may be used:

**–c** Complement the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal;

**–d** Delete all input characters in *string1*.

**–s** Squeeze all strings of repeated output characters that are in *string2* to single characters.

## EXAMPLE

The following example creates a list of all the words in *filename1* one per line in *filename2*, where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect '\' from the shell. 012 is the ASCII code for NEWLINE.

**tr –cs A–Za–z '\012'** <*filename1* >*filename2*

In the System V version, this would be specified as:

**tr –cs '[A–Z][a–z]' '[\012*]'** <*filename1* >*filename2*

**SEE ALSO**

**ed**(1), **expand**(1), **ascii**(7)

**BUGS**

Will not handle ASCII NUL in *string1* or *string2*. **tr** always deletes NUL from input.

## NAME

trace – trace system calls and signals

## SYNOPSIS

**trace** [ **–ct** ] [ **–o***filename* ] *command*

**trace** [ **–ct** ] [ **–o***filename* ] **–p** *pid*

## DESCRIPTION

**trace** runs the specified *command* until it exits. It intercepts the system calls and signals of a process. The name of the system call, its arguments and result are listed on the standard output.

Each line in the trace contains the system call name, followed by its arguments in parentheses and its result. Error returns (result = –1) have the error name and error message appended. Signals are printed as a signal name followed by the signal number. Arguments are printed according to their type. Structure pointers are always printed as hex addresses. Character pointers are dereferenced and printed as a quoted string. Non-printing characters in strings are represented by escape codes. Only the first 32 bytes of strings are printed; longer strings have two dots appended following the closing quote.

**The quick brown fox jumps over t ..**

Strings with more than 50% non-printing characters are assumed to contain binary data and are represented by a null string followed by two dots.

""..

## OPTIONS

| | |
|---|---|
| **–c** | Print a quick summary of system call and signal counts, instead of printing a full trace. |
| **–t** | Prefix each line of the trace with the time of day. |
| **–o***filename* | Write the trace output to the file *filename* rather than to the standard output. |
| **–p** *pid* | Attach to the process with the process ID *pid* and begin tracing. The trace may be terminated at any time by a keyboard interrupt signal (CTRL-C); **trace** will respond by detaching itself from the traced process leaving it to continue running. |

## EXAMPLES

```
example% trace date
gettimeofday (0x21474, 0x2147c) = 0
gettimeofday (0x21474, 0) = 0
gettimeofday (0xefffc78, 0x214ac) = 0
ioctl (1, 0x40067408, 0xefffa10) = –1 ENOTTY (Inappropriate ioctl for device)
fstat (1, 0xefffa30) = 0
getpagesize () = 8192
brk (0x27640) = 0
close (0) = 0
Thu Dec  4 14:16:36 PST 1986
write (1, "Thu Dec  4 14:16:36 PST 1986\n", 29) = 29
close (1) = 0
close (2) = 0
exit (0) = ?
example%
```

## SEE ALSO

**ptrace**(2)

**NOTES**

A process that has a system call trace applied to it with the **-p** option will receive a **SIGSTOP**. This signal may interrupt a system call that is not restartable. This may have an unpredictable effect on the process if the process takes no action to restart the system call.

**BUGS**

Programs that use the *setuid* bit do not have effective user ID privileges while being traced.

Child processes of a traced process are not traced.

A traced process ignores SIGSTOP.

A traced process runs slowly.

## NAME

traffic – SunView program to display Ethernet traffic

## SYNOPSIS

**traffic** [ **–h** *host* ] [ **–s** *subwindows* ]

## AVAILABILITY

This command is available when both the *Networking* and the *SunView User's* software options are installed. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**traffic** graphically displays ethernet traffic. It gets statistics from **etherd**(8C), running on machine *host*. The tool is divided into subwindows, each giving a different view of network traffic.

## OPTIONS

**–h** *host* Specify a host from which to get statistics. The default value of *host* is the machine that **traffic** is running on.

**–s** *subwindows* Specify the number of subwindows to display initially. The default value of *subwindows* is 1.

## SUBWINDOWS

To the right of each subwindow is a panel that selects what the subwindow is viewing. When *Size* is checked, than the size distribution of packets is displayed. *Proto* is for protocol, *Src* is for source of packet, and *Dst* is for destination of packet. Since it is not possible to show all possible sources, when *Src* is selected, only the 8 highest sources are displayed (and similarly for *Dst* ).

For each of these choices, the distribution is displayed by a histogram. The panel above each subwindow controls characteristics of the histograms. At the left of the panel is a shaded square, corresponding to one of the two shades of bars in the histogram. You can switch the shade by either clicking on the square with the left button, or bringing up a menu over the square with the right mouse button. When the light colored square is visible, then the slider in the center of the panel controls how often the light colored bars are updated. When the dark square is visible, then the slider refers to the dark bars of the histogram. To the right of the slider is a choice of *Abs* versus *Rel*. This selects whether the height of the histogram is *Absolute* in packets per second, or *Relative* in percent of total packets on the ethernet. Next in the panel are three small horizontal bars. When selected (that is, when a check mark appears to the left of the three bars), a horizontal grid appears on the histogram. Finally the button marked *Delete Me* will delete the subwindow.

The right hand panel also has a choice for *Load* Load is represented as a strip chart, rather than a histogram. The maximum value of the graph represents a load of 100%, that is 10 megabits per second on the ethernet. When *Load* is selected, there is only one slider, and no *Rel* versus *Abs* choice.

At the very top of the tool is a panel that contains filters, as well as a *Split* button that splits the tool and creates a new subwindow, and a *Quit* button that exits the tool. The filters apply to all the subwindows. When a filter is selected, a check mark appears to the left of the word *Filter*. There can be more than one filter active at the same time. The meaning of each filter is as follows. *Src* is a host or net, which can be specified either by name or address (similarly for *Dst*). *Proto* is an ip protocol, and can either be a name (such as *udp*, *icmp*) or a number. *Lnth* is either a packet length, or a range of lengths separated by a dash.

## SEE ALSO

**etherd**(8C)

## BUGS

If multiple copies of **traffic** are using the same copy of **etherd**, and one of them invokes a filter, then all the copies of **traffic** will be filtered.

## NAME

troff – typeset or format documents

## SYNOPSIS

**troff** [ **–abfiqtwz** ] [ **–m***package* ] [ **–n***N* ] [ **–o***pagelist* ] [ **–p***N* ] [ **–r***aN* ] [ **–s***N* ] [ *filenames* ] ...

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**troff** formats text in the *filenames*. For historical reasons, output goes to a CAT/4 phototypesetter attached to **/dev/cat**, but nobody uses a CAT/4 anymore. Ordinarily, postprocessing software converts output to a form that can be printed on newer typesetters or laser printers. Default font width tables correspond to Times Roman on PostScript™ printers. See also the **nroff**(1) manual page, which describes a formatter for typewriter-like devices.

Input to **troff** is expected to consist of text interspersed with formatting requests and macros. If no *filename* argument is present, **troff** reads standard input. A '–' as a *filename* argument indicates that standard input is to be read at that point in the list of input files; **troff** reads the files named ahead of the '–' in the arguments list, then text from the standard input, and then text from the files named after the '–'.

If the file **/etc/adm/tracct** is writable, **troff** keeps printer accounting records there. The integrity of that file may be secured by making **troff** a "set-user-ID" program (see **chmod**(1V) for details on the **setuid** permission bit.)

## OPTIONS

Options may appear in any order, but they all must appear before the first *filename*.

**–a** Send a printable ASCII approximation of the results to the standard output.

**–i** Read the standard input after the input files are exhausted.

**–q** Disable echoing during a **.rd** request.

**–t** Direct output to the standard output instead of the printer. Since this output is non-ASCII it is generally redirected to **lpr –t**.

**–m***package*
Prepend the macro file **/usr/lib/tmac/tmac.***package* to the input *filenames*. (Note that most references to macro packages include the leading "**m**" as part of the name; the **man**(7) macro package resides in **/usr/lib/tmac/tmac.an**).

**–n***N* Number first generated page *N*.

**–o***list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end.

**–r***aN* Set register *a* (one-character) to *N*.

Some options of **troff** only apply if you have a CAT/4 typesetter attached to your system. These options remain present for backward compatibility. However, this version of **troff** does not support this typesetter by default.

**–b** Report whether the typesetter is busy or available. No text processing is done.

**–f** Refrain from feeding paper out and stopping at the end of the print job on the typesetter.

**–w** Wait until typesetter is available, if currently busy.

**–z** Suppress all formatted output. Display only terminal messages produced by **.tm** requests and diagnostics.

**–pN** Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce elapsed time on the typesetter.

**–s***N* Stop the phototypesetter every *N* pages. **troff** produces a trailer so you can change cassettes; resume by pressing the typesetter's start button.

## FILES

| | |
|---|---|
| **/tmp/ta*** | temporary file |
| **/usr/lib/tmac/tmac.*** | standard macro files |
| **/usr/lib/term/*** | terminal driving tables for **nroff** |
| **/usr/lib/font/*** | font width tables for alternate mounted **troff** fonts |
| **/dev/cat** | phototypesetter |
| **/etc/adm/tracct** | accounting statistics for **/dev/cat** |

## SEE ALSO

**checknr**(1), **chmod**(1V), **col**(1V), **eqn**(1), **lpr**(1), **nroff**(1), **tbl**(1), **printcap**(5), **man**(7), **me**(7), **ms**(7), **lpd**(8)

*Formatting Documents*
*Using* `nroff` *and* `troff`

## DIAGNOSTICS

**No /dev/cat: try –t or –a**

The CAT/4 typesetter is not accessible from your machine. Combine the **–t** option of **troff** with the **–t** option of **lpr**(1) to get output on a laser printer or typesetter. For information on how to inform **lpd**(8) of a PostScript printer attached to a remote host, see **printcap**(5).

## NOTES

**troff** is not 8-bit clean because it is by design based on 7-bit ASCII.

**NAME**

true, false – provide truth values

**SYNOPSIS**

**true**

**false**

**DESCRIPTION**

**true** and **false** are usually used in a Bourne shell script. They test for the appropriate status "true" or "false" before running (or failing to run) a list of commands.

**EXAMPLE**

The following Bourne shell script will be executed while the case status is "true".

```
while true
do
        command list
done
```

**SEE ALSO**

**csh**(1), **sh**(1)

**EXIT STATUS**

**true** has exit status 0.

**false** has exit status 1.

## NAME

tset, reset – establish or restore terminal characteristics

## SYNOPSIS

**tset** [ **–InQrsS** ] [ **–e***c* ] [ **–k***c* ] [ **–m** [*port*–ID [ *baudrate*] : *type* ] ... ] [*type*]

**reset** [ – ] [ **–e***c* ] [ **–I** ] [ **–k***c* ] [ **–n** ] [ **–Q** ] [ **–r** ] [ **–s** ] [ **–S** ]
**–m** [ *indent* ] [ *test baudrate* ]: *type* ] ... [ *type* ]

## DESCRIPTION

**tset** sets up your terminal, typically when you first log in. It does terminal dependent processing such as setting erase and kill characters, setting or resetting delays, sending any sequences needed to properly initialized the terminal, and the like. **tset** first determines the *type* of terminal involved, and then does necessary initializations and mode settings. The type of terminal attached to each port is specified in the **/etc/ttytab** database. Type names for terminals may be found in the **termcap**(5) database. If a port is not wired permanently to a specific terminal (not hardwired) it is given an appropriate generic identifier such as **dialup**.

**reset** clears the terminal settings by turning off CBREAK and RAW modes, output delays and parity checking, turns on NEWLINE translation, echo and TAB expansion, and restores undefined special characters to their default state. It then sets the modes as usual, based on the terminal type (which will probably override some of the above). (See **stty**(1V) for more information.) All arguments to **tset** may be used with **reset**. **reset** also uses the **rs=** and **rf=** (reset string and file) instead of the initialization string and file from **/etc/termcap**. This is useful after a program dies and leaves the terminal in a funny state. Often in this situation, characters will not echo as you type them. You may have to type '**LINEFEED reset LINEFEED**' since '**RETURN**' may not work.

When no arguments are specified, **tset** reads the terminal type from the **TERM** environment variable and re-initializes the terminal, and performs initialization of mode, environment and other options at login time to determine the terminal type and set up terminal modes.

When used in a startup script (**.profile** for **sh**(1) users or **.login** for **csh**(1) users) it is desirable to give information about the type of terminal you will usually use on ports that are not hardwired. These ports are identified in **/etc/ttyab** as **dialup** or **plugboard**, etc. Any of the alternate generic names given in **/etc/termcap** may be used for the identifier. Refer to the **–m** option under OPTIONS for more information. If no mapping applies and a final *type* option, not preceded by a **–m**, is given on the command line then that type is used; otherwise the type found in the **/etc/ttytab** database is used as the terminal type. This should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by **tset**, and information about the terminal's capabilities, to a shell's environment. This can be done using the –, **–s**, or **–S** options. (Refer to OPTIONS for more information.)

For the Bourne shell, put this command in your **.profile** file:

**eval `tset –s** *options...*`

or using the C shell, put this command in your **.login** file:

**eval `tset –s** *options...*`

With the C shell, it is also convenient to make an alias in your **.cshrc** file:

**alias tset 'eval `tset –s \!*`'**

This also allows the command:

**tset 2621**

to be invoked at any time to set the terminal and environment. *Note to Bourne Shell users:* It is *not* possible to get this aliasing effect with a shell script, because shell scripts cannot set the environment of their parent. If a process could set its parent's environment, none of this nonsense would be necessary in the first place.

Once the terminal type is known, **tset** sets the terminal driver mode. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. TAB and NEWLINE expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is '#', the erase character is changed as if **-e** had been used.

**OPTIONS**

**–** The name of the terminal finally decided upon is output on the standard output. This is intended to be captured by the shell and placed in the **TERM** environment variable.

**–e***c* Set the erase character to be the named character *c* on all terminals. Default is the backspace key on the keyboard, usually **^H** (CTRL-H). The character *c* can either be typed directly, or entered using the circumflex-character notation used here.

**–i***c* Set the interrupt character to be the named character *c* on all terminals. Default is **^C** (CTRL-C). The character *c* can either be typed directly, or entered using the circumflex-character notation used here.

**–I** Suppress transmitting terminal-initialization strings.

**–k***c* Set the line kill character to be the named character *c* on all terminals. Default is **^U** (CTRL-U). The kill character is left alone if **–k** is not specified. Control characters can be specified by prefixing the alphabetical character with a circumflex (as in CTRL-U) instead of entering the actual control key itself. This allows you to specify control keys that are currently assigned.

**–n** Specify that the new tty driver modes should be initialized for this terminal. Probably useless since **stty new** is the default.

**–Q** Suppress printing the '**Erase set to**' and '**Kill set to**' messages.

**–r** In addition to other actions, reports the terminal type.

**–s** Output commands to set and export **TERM** and **TERMCAP**. This can be used with

```
set noglob
eval `tset –s ...`
unset noglob
```

to bring the terminal information into the environment. Doing so makes programs such as **vi**(1) start up faster. If the **SHELL** environment variable ends with **csh**, C shell commands are output, otherwise Bourne shell commands are output.

**–S** Similar to the **–s** option, but produces two strings containing suitable values for the (environment) variables **TERM** and **TERMCAP**, respectively, and can be used as follows:

```
set noglob
set t=(`tset –S ...`)
setenv TERM $t[1]
setenv TERMCAP "$t[2]"
unset t
unset noglob
```

Since **–s** loads these values, its use is preferred. If the **SHELL** environment variable does not end with **csh**, **–S** produces the same Bourne shell commands that **–s** does.

**–m** [*port*-ID[*baudrate*]:*type*] ...

Specify (map) a terminal type when connected to a generic port (such as *dialup* or *plugboard*) identified by *port-ID*. The *baudrate* argument can be used to check the baudrate of the port and set the terminal type accordingly. The target rate is prefixed by any combination of the following operators to specify the conditions under which the mapping is made:

| | |
|---|---|
| **>** | Greater than |
| **@** | Equals or "at" |
| **<** | Less than |
| **!** | It is not the case that (negates the above operators) |
| **?** | Prompt for the terminal type. If no response is given, then *type* is selected by default. |

In the following example, the terminal type is set to **adm3a** if the port is a dialup with a speed of greater than 300 or to *dw2* if the port is a dialup at 300 baud or less. In the third case, the question mark preceding the terminal type indicates that the user is to verify the type desired. A null response indicates that the named type is correct. Otherwise, the user's response is taken to be the type desired.

**tset –m 'dialup>300:adm3a' –m 'dialup:dw2' –m 'plugboard:?adm3a'**

To prevent interpretation as metacharacters, the entire argument to **–m** should be enclosed in single quotes. When using the C shell, exclamation points should be preceded by a backslash (\).

**EXAMPLES**

These examples all use the '–' option. A typical use of **tset** in a **.profile** or **.login** will also use the **–e** and **–k** options, and often the **–n** or **–Q** options as well. These options have been omitted here to keep the examples short.

To select a 2621, you might put the following sequence of commands in your **.login** file (or **.profile** for Bourne shell users).

```
set noglob
eval `tset –s 2621`
unset noglob
```

If you have an h19 at home which you dial up on, but your office terminal is hardwired and known in **/etc/ttytab**, you might use:

```
set noglob
eval `tset –s –m dialup:h19`
unset noglob
```

If you have a switch which connects to various ports (making it impractical to identify which port you may be connected to), and use various terminals from time to time, you can select from among those terminals according to the *speed* or baud rate. In the example below, **tset** will prompt you for a terminal type if the baud rate is greater than 1200 (say, 9600 for a terminal connected by an RS-232 line), and use a Wyse 50 by default. If the baud rate is less than or equal to 1200, it will select a 2621. Note the placement of the question mark, and the quotes to protect the > and ? from interpretation by the shell.

```
set noglob
eval `tset –s –m 'switch>1200:?wy' –m 'switch<=1200:2621'`
unset noglob
```

All of the above entries will fall back on the terminal type specified in **/etc/ttytab** if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals, and the terminal you use most often is an adm3a.

```
set noglob
eval `tset –s ?adm3a`
unset noglob
```

If the file **/etc/ttytab** is not properly set up and you want to make the selection based only on the baud rate, you might use the following:

```
set noglob
eval `tset -s -m '>1200:wy' 2621`
unset noglob
```

The following example quietly sets the erase character to BACKSPACE, and kill to CTRL-U. If the port is switched, it selects a Concept 100 for speeds less than or equal to 1200, and asks for the terminal type otherwise (the default in this case is a Wyse 50). If the port is a direct dialup, it selects Concept 100 as the terminal type. If logging in over the TCP/IP, the terminal type selected is a Datamedia 2500 terminal or emulator. (Note the backslash escaping the NEWLINE at the end of the first line in the example.)

```
set noglob
eval `tset -e -k^U -Q -s -m 'switch<=1200:concept100' -m \        'switch:?wy' -m
dialup:concept100 -m arpanet:dm2500`
unset noglob
```

FILES

| | |
|---|---|
| **/etc/ttytab** | port name to terminal type mapping database |
| **/etc/termcap** | terminal capability database |
| **/usr/share/lib/tabset/*** | TAB setting sequences for various terminals. Pointed to by **termcap** entries. |
| **.login** | |
| **.profile** | |

SEE ALSO

**csh**(1), **sh**(1), **stty**(1V), **vi**(1), **environ**(5V), **termcap**(5), **ttytab**(5)

NOTES

Once the terminal's size has been initialized, further invocations of **tset** will not affect it. To correct this, do the following:

```
example% stty rows 0 colums 0
```

and then run **tset** normally.

BUGS

The **tset** command is one of the first commands a user must master when getting started on a UNIX system. Unfortunately, it is one of the most complex, largely because of the extra effort the user must go through to get the environment of the login shell set. Something needs to be done to make all this simpler, either the **login** program should do this stuff, or a default shell alias should be made, or a way to set the environment of the parent should exist.

This program cannot intuit personal choices for erase, interrupt and line kill characters, so it leaves these set to the local system standards.

It could well be argued that the shell should be responsible for ensuring that the terminal remains in a sane state; this would eliminate the need for the **reset** program.

## NAME

tsort – topological sort

## SYNOPSIS

**tsort** [ *filename* ]

## DESCRIPTION

**tsort** produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *filename*. If no *filename* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by SPACE characters. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

## SEE ALSO

**lorder**(1)

## BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

**NAME**

tty – display the name of the terminal

**SYNOPSIS**

**tty** [ –s ]

**DESCRIPTION**

**tty** prints the pathname of the user's terminal unless the –s (silent) option is given. In either case, the exit value is zero if the standard input is a terminal, and one if it is not.

**OPTIONS**

–s Silent. Does not print the pathname of the user's terminal.

## NAME

ul – do underlining

## SYNOPSIS

**ul** [ **–i** ] [ **–t** *terminal* ] [ *filename*... ]

## DESCRIPTION

**ul** reads the named *filename*s (or the standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable **TERM**. **ul** uses the **/etc/termcap** file to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, **ul** degenerates to **cat**(1V). If the terminal cannot underline, underlining is ignored.

## OPTIONS

**–t** *terminal*

Override the terminal kind specified in the environment. If the terminal cannot underline, underlining is ignored.

**–i** Indicate underlining by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an **nroff**(1) output stream on a CRT-terminal.

## FILES

**/etc/termcap**

## SEE ALSO

**cat**(1V), **colcrt**(1), **man**(1), **nroff**(1)

## BUGS

**nroff** usually generates a series of backspaces and underlines intermixed with the text to indicate underlining. **ul** makes attempt to optimize the backward motion.

**NAME**

uname – display the name of the current system

**SYNOPSIS**

**uname** [ **–mnrsva** ]

**DESCRIPTION**

**uname** prints information about the current system on the standard output. If no options are specified, **uname** prints the current system's name.

**OPTIONS**

**–m** Print the machine hardware name.

**–n** Print the nodename. The nodename may be a name that the system is known by to a communications network.

**–r** Print the operating system release.

**–s** Print the system name (default).

**–v** Print the operating system version.

**–a** Print all the above information.

**SEE ALSO**

**uname**(2V)

## NAME

unifdef – resolve and remove ifdef'ed lines from cpp input

## SYNOPSIS

**unifdef** [ **–c l t** ] [ **–D***name* ] [ **–U***name* ] [ **–iD***name* ] [ **–iU***name* ] ... [ *filename* ]

## DESCRIPTION

**unifdef** removes **ifdef**ed lines from a file while otherwise leaving the file alone. It is smart enough to deal with the nested **ifdef**s, comments, single and double quotes of C syntax, but it does not do any including or interpretation of macros. Neither does it strip out comments, though it recognizes and ignores them. You specify which symbols you want defined with **–D** options, and which you want undefined with **–U** options. Lines within those **ifdef**s will be copied to the output, or removed, as appropriate. Any **ifdef**, **ifndef**, **else**, and **endif** lines associated with *filename* will also be removed.

**ifdef**s involving symbols you do not specify are untouched and copied out along with their associated **ifdef**, **else**, and **endif** lines.

If an **ifdef***X* occurs nested inside another **ifdef***X*, then the inside **ifdef** is treated as if it were an unrecognized symbol. If the same symbol appears in more than one argument, only the first occurrence is significant.

**unifdef** copies its output to the standard output and will take its input from the standard input if no *filename* argument is given.

## OPTIONS

**–c** Complement the normal operation. Lines that would have been removed or blanked are retained, and vice versa.

**–l** Replace "lines removed" lines with blank lines.

**–t** Plain text option. **unifdef** refrains from attempting to recognize comments and single and double quotes.

**–iD***name*

Ignore, but print out, lines associated with the defined symbol *filename*. If you use **ifdef**s to delimit non-C lines, such as comments or code which is under construction, then you must tell **unifdef** which symbols are used for that purpose so that it won't try to parse for quotes and comments within them.

**–iU***name*

Ignore, but print out, lines associated with the undefined symbol *filename*.

## SEE ALSO

**cpp**(1), **diff**(1)

## DIAGNOSTICS

**Premature EOF**

Inappropriate **else** or **endif**.

Exit status is 0 if output is exact copy of input, 1 if not, 2 if trouble.

## BUGS

Does not know how to deal with **cpp**(1) constructs such as

```
#if     defined(X) || defined(Y)
```

## NAME

uniq – remove or report adjacent duplicate lines

## SYNOPSIS

**uniq** [ **–cdu** [ +|–*n* ] [ *inputfile* [ *outputfile* ] ]

## DESCRIPTION

**uniq** reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note: repeated lines must be adjacent in order to be found; see **sort**(1V).

## OPTIONS

**–c** Supersede **–u** and **–d** and generate an output report in default style but with each line preceded by a count of the number of times it occurred.

The normal output of **uniq** is the union of the **–u** and **–d** options.

**–d** Write one copy of just the repeated lines.

**–u** Copy only those lines which are *not* repeated in the original file.

The *n* arguments specify skipping an initial portion of each line in the comparison:

*+n* The first *n* characters are ignored. Fields are skipped before characters.

*–n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-SPACE, non-TAB characters separated by SPACE and TAB characters from its neighbors.

## SEE ALSO

**comm**(1), **sort**(1V)

## NAME

units – conversion program

## SYNOPSIS

**units**

## DESCRIPTION

**units** converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

**units** only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

| | |
|---|---|
| **pi** | Ratio of circumference to diameter, |
| **c** | Speed of light, |
| **e** | Charge on an electron, |
| **g** | Acceleration of gravity, |
| **force** | Same as **g**, |
| **mole** | Avogadro's number, |
| **water** | Pressure head per unit height of water, |
| **au** | Astronomical unit. |

**pound** is not recognized as a unit of mass; **lb** is. **pound** refers to a British pound. Compound names are run together (for instance, **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. Currency is denoted **belgiumfranc**, **britainpound**, ... For a complete list of units, type:

**cat /usr/lib/units**

## FILES

**/usr/lib/units**

## BUGS

Do not base your financial plans on the currency conversions.

## NAME

unix2dos – convert text file from ISO format to DOS format

## SYNOPSIS

**unix2dos** [ **–ascii** ] [ **–iso** ] [ **–7** ] *originalfile convertedfile*

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

**unix2dos** converts ISO standard characters to the corresponding characters in the DOS extended character set.

This command may be invoked from either DOS or SunOS. However, the filenames must conform to the conventions of the environment in which the command is invoked.

If the original file and the converted file are the same, **unix2dos** will rewrite the original file after converting it.

## OPTIONS

**–ascii** Adds carriage returns and converts end of file characters in SunOS format text files to conform to DOS requirements.

**–iso** This is the default. Converts ISO standard characters to the corresponding character in the DOS extended character set.

**–7** Convert 8 bit SunOS characters to 7 bit DOS characters.

## DIAGNOSTICS

**File** *filename* **not found, or no read permission**

The input file you specified does not exist, or you do not have read permission (check with the SunOS command **ls –l**).

**Bad output filename** *filename*, **or no write permission**

The output file you specified is either invalid, or you do not have write permission for that file or the directory that contains it. Check also that the drive or diskette is not write-protected.

**Error while writing to temporary file**

An error occurred while converting your file, possibly because there is not enough space on the current drive. Check the amount of space on the current drive using the DIR command. Also be certain that the default diskette or drive is write-enabled (not write-protected). Note that when this error occurs, the original file remains intact.

**Could not rename tmpfile to** *filename*.
**Translated tmpfile name =** *filename*.

The program could not perform the final step in converting your file. Your converted file is stored under the name indicated on the second line of this message.

## SEE ALSO

**dos**(1), **dos2unix**(1)

*Sun386i Advanced Skills*
*Sun MS-DOS Reference Manual*

## NAME

unload, unloadc – unload Application SunOS or Developer's Toolkit optional clusters

## SYNOPSIS

**unload** *filename*...

**unloadc** *cluster*...

## AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*.x* release or earlier. Not a SunOS 4.1 release feature.

## DESCRIPTION

**unload** unloads the Application SunOS or Developer's Toolkit clusters that contain the specified file arguments. **unloadc** unloads the specified Application SunOS or Developer's Toolkit clusters.

Without arguments, **unload** and **unloadc** display a summary of the clusters in the Application SunOS and Developer's Toolkit, including the load state and size of each cluster.

## EXAMPLES

To unload the cluster that contains the **spell** command:

```
example% unload spell
About to unload the spellcheck cluster, confirm (y/n): y
Unloading the spellcheck cluster ...
The spellcheck cluster has been unloaded.
space used by clusters: 5358K bytes
total space remaining: 20962K bytes
example%
```

To display a summary of the clusters in the Application SunOS and Developer's Toolkit:

```
example% unload
Application SunOS Optional Clusters:
  available   cluster          size (bytes)
  ---------   -------          ----
    yes       accounting       265K
    no        advanced_admin   501K
    ...

Developer's Toolkit Optional Clusters:
  available   cluster          size (bytes)
  ---------   -------          ----
    no        base_devel       6907K
    ...

space used by clusters: 6021K bytes
total space remaining: 20432K bytes
```

## FILES

**/export/cluster/sun386.sunos***release***/appl**

where Application SunOS clusters are loaded (or mounted). *release* is the current release of the operating system, for example, 4.0.2.

**/export/cluster/sun386.sunos*release*/devel**
where Developer's Toolkit clusters are loaded (or mounted). *release* is the current release of the operating system, for example, 4.0.2.

**/usr/lib/load/***
data files

**SEE ALSO**

**cluster**(1), **load**(1), **toc**(5)
*Sun386i Setup and Maintenance*

**DIAGNOSTICS**

**The file *filename* is not in any of the optional software clusters.**
The specified file is not part of the Application SunOS or Developer's Toolkit.

**The cluster *cluster* is not loaded.**
The specified cluster is not loaded on disk.

**There is no *cluster* cluster.**
The specified cluster is not part of the Application SunOS or Developer's Toolkit.

**The Application SunOS (and/or) Developer's Toolkit are mounted.**
The Application SunOS or Developer's Toolkit or both are mounted across the network and can not be loaded or unloaded.

**NAME**

unwhiteout – remove a TFS whiteout entry

**SYNOPSIS**

**unwhiteout** *filename* ...

**DESCRIPTION**

For each *filename* that is a whiteout entry, **unwhiteout** removes that entry from the containing directory. These whiteout entries are created by the translucent file service. See **tfs**(4S).

**SEE ALSO**

**lsw**(1), **tfs**(4S), **mount_tfs**(8), **tfsd**(8)

**NAME**

uptime – show how long the system has been up

**SYNOPSIS**

**uptime**

**DESCRIPTION**

**uptime** prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a w(1) command.

**EXAMPLE**

```
example% uptime
6:47am  up 6 days, 16:38,  1 users,  load average: 0.69, 0.28, 0.17
example%
```

**FILES**

| | |
|---|---|
| **/vmunix** | system namelist |

**SEE ALSO**

w(1)

**NAME**

users – display a compact list of users logged in

**SYNOPSIS**

**users**

**DESCRIPTION**

**users** lists the login names of the users currently on the system in a compact, one-line format:

```
example% users
paul george ringo
example%
```

**FILES**

**/etc/utmp**

**SEE ALSO**

**who**(1)

## NAME

ustar – process tape archives

## SYNOPSIS

**/usr/5bin/ustar –c** [ **bfvw** ] *device block filename* ...
**/usr/5bin/ustar –r** [ **bvw** ] *device block* [ *filename* ... ]
**/usr/5bin/ustar –t** [ **fv** ] *device*
**/usr/5bin/ustar –u** [ **bvw** ] *device block*
**/usr/5bin/ustar –x** [ **flmovw** ] *device* [ *filename* ... ]

## AVAILABILITY

**ustar** is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**ustar** reads and writes archive files which conform to the **Archive/Interchange File Format** specified in *IEEE Std. 1003.1-1988*.

If the files exist, their modes are not changed except as described above. The owner, group and modification time are restored if possible. If no *filename* argument is given, the entire contents of the archive is extracted. Note: if several files with the same name are in the archive, the last one will overwrite all earlier ones.

## OPTIONS

**–c** Create a new archive; writing begins at the beginning of the archive, instead of after the last file.

**–r** Write names files to the end of the archive.

**–t** List the names of all of the files in the archive.

**–u** Add named files to the archive if they are not already there, or if they have been modified since last written into the archive. This implies the **–r** option.

**–x** Extracts named files from the archive. If a named file matches a directory whose contents had been written onto the archive, that directory is recursively extracted. If a named file in the archive does not exist on the system, the file is create with the same mode as the one in the archive, except that the set-user-id and get-group-id modes are not set unless the user has appropriate privileges.

**–b** Use the next argument on the command line as the blocking factor for tape records. The default is 1; the maximum is 20. This option should only be used with raw magnetic tape archives. Normally, the block size is determined automatically when reading tapes.

**–f** Use the next argument on the command line as the name of the archive instead of the default, which is usually a tape drive. If '–' is specified as a filename **ustar** writes to the standard output or reads from the standard input, whichever is appropriate for the options given. Thus, **ustar** can be used as the head or tail of a pipeline.

**–l** Report if **ustar** cannot resolve all of the links to the files being archived. If **–l** is not specified, no error messages are written to the standard output. This modifier is only valid with the **–c**, **–r** and **–u** options.

**–m** Do not restore the modification times. The modification time of the file will be the time of extraction. This modifier is invalid with the **–t** option.

**–o** Extracted files take on the user and group identifier of the user running the program rather than those on the archive. This modifier is only valid with the **–x** option.

**–v** Verbose. Print the name of each file it processes, preceded by the option letter. With the **–t** option, **–v** gives more information about the archive entries than just the name.

**–w** Print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with y is given, the action is performed. Any other input implies no and the action is not performed. This modifier is invalid with the **–t** option.

**FILES**

**/dev/tty** used to prompt the user for information when the **–i** or **–y** options are specified

**SEE ALSO**

**cpio**(1), **dd**(1), **find**(1), **pax**(1V), **paxcpio**(1V), **cpio**(5), **tar**(5)

**AUTHOR**

Mark H. Colburn
NAPS International
117 Mackubin Street, Suite 1
St. Paul, MN 55102

Sponsored by **The USENIX Association** for public distribution.

## NAME

uucp, uulog, uuname – system to system copy

## SYNOPSIS

**uucp** [ **–cCdfjmr** ] [ **–g***grade* ] [ **–n***username* ] [ **–x** *debug_level* ] *source-file* ... *destination-file*

**uulog** [ **–x** ] [ **–f***system* ] [ **–s***system* ] [ *–number* ]

**uuname** [ **–cl** ]

## AVAILABILITY

These commands are available with the *uucp* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

### uucp

**uucp** copies each *source-file* to the named *destination-file*. A filename may be a path name on your machine, or may have the form

*system-name*!*pathname*

where *system-name* is taken from a list of system names that **uucp** knows about. The *system-name* may also be a list of names such as

*system-name*!*system-name*!...!*system-name*!*pathname*

in which case an attempt is made to send the file by way of the specified route, to the destination. See WARNINGS and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to foward information (see WARNINGS below for restrictions).

The shell metacharacters **?**, *****, and **[ ]** appearing in the pathname part will be expanded on the appropriate system.

Path names may be one of:

- a full pathname;
- a pathname preceded by *~username/*; *username* is interpreted as a username on the specified system and is replaced by that user's login directory on that system;
- a pathname preceded by *~/destination/*; the *~/* is replaced by the "public UUCP" directory on the remote machine Note: this destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example **~/dan/** as the destination will make the directory **/usr/spool/uucppublic/dan** if it does not exist and put the requested file(s) in that directory;
- a partial pathname, which is prefixed by the pathname of current directory.

If the result is an erroneous pathname for the remote system, the copy will fail. If the *destination-file* is a directory, the last component of the *source-file* name is used.

**uucp** preserves execute permissions across the transmission and gives 0666 read and write permissions (see **chmod**(2V)).

### uulog

**uulog** queries the log file **/var/spool/uucp/.Log/uucico/***system* of **uucp** transactions for system *system*, or the log file **/var/spool/uucp/.Log/uuxqt/***system* of **uux**(1C) transactions for system *system*.

### uuname

**uuname** lists the UUCP names of systems that can be accessed using **uucp**.

**OPTIONS**

**uucp**

**–c** Use the source file when copying out rather than copying the file to the spool directory. This is the default.

**–C** Make a copy of outgoing files in the UUCP spool directory, rather than copying the source file directly to the target system. This lets you remove the source file after issuing the **uucp** command.

**–d** Make all necessary directories for the file copy. This is the default.

**–f** Do not make intermediate directories for the file copy.

**–j** Output the job identification ASCII string on the standard output. This job identification can be used by **uustat**(1C) to obtain the status or terminate a job.

**–m** Send mail to the requester when the copy is complete.

**–r** Do not start **uucico**(8C), just queue the job.

**–g***grade*

*grade* is a single letter or number, from **0** to **9**, **A** to **Z**, or **a** to **z**; **0** is the highest grade, and **z** is the lowest grade. Lower grades will cause the job to be transmitted earlier during a particular conversation. The default *grade* is **n**. By way of comparison, **uux**(1C) defaults to **A**; mail is usually sent at grade **C**.

**–n***username*

Notify *username* on the remote system (that is, send *username* mail) that a file was sent.

**–x** *debug_level*

Produce debugging output on the standard output. *debug_level* is a number between 0 and 9; higher numbers give more detailed information. 5, 7, and 9 are good numbers to try; they give increasing amounts of detail.

**uulog**

**–x** Look in the **uuxqt**(8C) log file for the given system.

**–f***system*

Does a '**tail –f**' of the file transfer log for *system*. You must hit BREAK to exit this function.

**–s***system*

Print information about work involving system *system*.

*–number*

Indicate that a **tail** command of *number* lines should be executed.

**uuname**

**–c** Display the names of systems known to **cu**(1C). The two lists are the same, unless your machine is using different **Systems** files for **cu** and **uucp**. See the **Sysfiles** file.

**–l** Display the local system name.

**FILES**

| | |
|---|---|
| **/var/spool/uucp** | spool directories |
| **/var/spool/uucppublic** | public directory for receiving and sending |
| **/etc/uucp/*** | other data files |
| **/usr/lib/uucp/*** | other program files |

**SEE ALSO**

**mail**(1), **uustat**(1C), **uux**(1C), **chmod**(2V), **uucico**(8C), **uuxqt**(8C)

*System and Network Administration*

**WARNINGS**

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to ~/).

All files received by **uucp** will be owned by the user ID **uucp**.

The **–m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters **?**, *, and [ ] will not activate the **–m** option.

The forwarding of files through other systems may not be compatible with other versions of UUCP. If forwarding is used, all systems in the route must have the same version of UUCP.

When invoking **uucp** from **csh**(1), the **!** character must be prefixed by the \ escape to inhibit **csh**'s history mechanism. Quotes are not sufficient.

**BUGS**

Protected files and files that are in protected directories that are owned by the requestor can be sent by **uucp**. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

## NAME

uuencode, uudecode – encode a binary file, or decode its ASCII representation

## SYNOPSIS

**uuencode** [ *source-file* ] *file-label*

**uudecode** [ *encoded-file* ]

## DESCRIPTION

**uuencode** converts a binary file into an ASCII-encoded representation that can be sent using **mail**(1). It encodes the contents of *source-file*, or the standard input if no *source-file* argument is given. The *file-label* argument is required. It is included in the encoded file's header as the name of the file into which **uudecode** is to place the binary (decoded) data. **uuencode** also includes the ownership and permission modes of *source-file*, so that *file-label* is recreated with those same ownership and permission modes.

If the remote host is a UNIX system with the **sendmail**(8) mail-message delivery daemon, you can pipe the output of **uuencode** through **mail**(1) to the recipient named **decode** on the remote host. This recipient is typically an alias for the **uudecode** program (see **aliases**(5) for details), which allows a binary file to be decoded (extracted) from a mail message automatically. If this alias is absent on a particular host, the encoded file can be mailed to a user, who can run it through **uudecode** manually.

*uudecode* reads an *encoded-file*, strips off any leading and trailing lines added by mailer programs, and recreates the original binary data with the filename and the mode and owner specified in the header.

The encoded file is an ordinary ASCII text file; it can be edited by any text editor. But it is best only to change the mode or file-label in the header to avoid corrupting the decoded binary.

## SEE ALSO

**mail**(1), **uucp**(1C), **uusend**(1C), **uux**(1C), **aliases**(5), **uuencode**(5), **sendmail**(8)

## BUGS

The encoded file's size is expanded by 35% (3 bytes become 4, plus control information), causing it to take longer to transmit than the equivalent binary.

The user on the remote system who is invoking **uudecode** (typically **uucp**) must have write permission on the file specified in the *file-label*.

Since both **uuencode** and **uudecode** run with user ID set to **uucp**, **uudecode** can fail with "permission denied" when attempted in a directory that does not have write permission allowed for "other."

## NAME

uusend – send a file to a remote host

## SYNOPSIS

**uusend** [ **–fr** ] [ **–m** *mode* ] *source-file system-name1*!*system-name2*!...!*destination-file*

## AVAILABILITY

This command is available with the *uucp* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**uusend** sends the file, *source-file*, to a given location, *destination-file*, on a remote system. The system need not be directly connected to the local system, but a chain of **uucp**(1C) links must connect the two systems. If *destination-file* is a directory, the last component of the *source-file* name is used.

The source file can be '–', meaning to use the standard input. These options are primarily intended for internal use of **uusend**.

The destination file can include the ~*username* or ~/ syntax.

## OPTIONS

**–f** Do not make intermediate directories for sending the file.

**–r** Do not start **uucico**, just queue the job.

**–m** *mode*

Take the mode of the file on the remote end from the octal number specified as *mode*. The mode of the input file is used if the **–m** option is not specified.

## SEE ALSO

**uucp**(1C), **uuencode**(1C), **uux**(1C)

## BUGS

This command should not exist, since **uucp** should handle it.

All systems along the line must have the **uusend** command available and allow remote execution of it.

Some UUCP systems have a bug where binary files cannot be the input to a **uux** command. If this bug exists in any system along the line, the file will show up severely corrupted.

## NAME

uustat – UUCP status inquiry and job control

## SYNOPSIS

**uustat** **–a** |**–m** |**–p** |**–q** |**–k***jobid* |**–r***jobid*

**uustat** [ **–s***system* ] [ **–u***user* ]

## AVAILABILITY

This command is available with the *uucp* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**uustat** displays the status of, or cancels, previously specified **uucp**(1C) commands. It also reports the status of **uucp** connections to other systems. When no options are given, **uustat** displays the status of all **uucp** requests issued by the current user.

## OPTIONS

Only one of the following options can be specified at a time:

**–a** Output all jobs in queue.

**–m** Report the status of accessibility of all machines.

**–p** Execute a **ps** for all the PIDs listed in the lock files.

**–q** List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in parentheses next to the number of **C.** or **X.** files, it is the age in days of the oldest **C./X.** file for that system. The **Retry** field represents the number of hours until the next possible call. The **Count** is the number of failure attempts. For systems with a moderate number of outstanding jobs, this could take 30 seconds or more to execute. An example of the output from **–q** is:

```
eagle      3C   04/07–11:07  NO DEVICES AVAILABLE
mh3bs3     2C   07/07–10:42  SUCCESSFUL
```

This indicates the number of command files that are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

**–k***jobid* Kill the UUCP request with job identification of *jobid*. You must either own the job to be killed, or be the super-user.

**–r***jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the next limit imposed by the daemon.

The following options can be specified separately or together:

**–s***system*
Report the status of all UUCP requests for remote system *system*.

**–u***user* Report the status of all UUCP requests issued by *user*.

Output for both the **-s** and **-u** options has the following format:

```
eaglen0000  4/07-11:01:03      (POLL)
eagleN1bd7  4/07-11:07         S       eagle  dan  522 /usr/dan/A
eagleC1bd8  4/07-11:07         S       eagle  dan  59 D.3b2al2ce4924
            4/07-11:07         S       eagle  dan  rmail mike
```

The first field is the job ID of the job. This is followed by the date and time. The next field is either an **S** or **R** depending on whether the job is to send or request a file. This is followed by the user ID of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution request, the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user, or an internal name created for data files associated with remote executions (**rmail** in this example).

**FILES**

**/var/spool/uucp/*** UUCP spool directories

**SEE ALSO**

**uucp**(1C)

## NAME

uuto, uupick – public system-to-system file copy

## SYNOPSIS

**uuto** [ **–mp** ] *source-file ... destination*

**uupick** [ **–s** *system* ]

## AVAILABILITY

This command is available with the *uucp* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

### uuto

**uuto** sends *source-files* to *destination*. **uuto** uses the **uucp**(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

*system-name*!*username*

where *system-name* is taken from a list of system names that **uucp** knows about (see **uucp**(1C)). *username* is the username of someone on the specified system.

The files (or sub-trees if directories are specified) are sent to the "public UUCP" directory on the remote machine *system-name*. Normally, this directory is **/usr/spool/uucppublic**. Specifically the files are sent to

**PUBDIR/receive**/*username*/*mysystem*/*file*

where **PUBDIR** is the "public UUCP" directory on *system-name*, *mysystem* is the system from which the files are sent, and *file* is the file being sent.

The destined recipient is notified by **mail**(1) of the arrival of files.

### uupick

**uupick** accepts or rejects the files transmitted to the user. Specifically, **uupick** searches the "public UUCP" directory on the local machine for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

**from** *system*: [**file** *file-name*] [ **dir** *dirname*] ?

**uupick** then reads a line from the standard input to determine the disposition of the file:

| | |
|---|---|
| <NEWLINE> | Go on to next entry. |
| **d** | Delete the entry. |
| **m** [ *dir* ] | Move the entry to named directory *dir*. If *dir* is not specified as a complete path name (in which **$HOME** is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory. |
| **a** [ *dir* ] | Same as **m** except moving all the files sent from *system*. |
| **p** | Print the content of the file. |
| **q** | Stop. |
| EOT (CTRL-D) | Same as **q**. |
| **!***command* | Escape to the shell to do *command*. |
| * | Print a command summary. |

**OPTIONS**

**uuto**

**-m** Send mail to the sender when the copy is complete.

**-p** Copy the source file into the spool directory before transmission.

**uupick**

**-s** *system*

Search only the "public UUCP" directory on the local machine for files sent from *system*.

**FILES**

**/usr/spool/uucppublic** public UUCP directory

**SEE ALSO**

**mail**(1), **uucp**(1C), **uustat**(1C), **uux**(1C), **uucleanup**(8C)

**WARNINGS**

In order to send files that begin with a dot (such as, **.profile**) the files must by qualified with a dot. For example: **.profile**, **.prof***, and **.profil?** are correct, whereas ***prof*** and **?profile** are incorrect.

## NAME

uux – remote system command execution

## SYNOPSIS

**uux** [ – ] [ **–bcCjnprz** ] [ **–a***name* ] [ **–g***grade* ] [ **–x** *debug_level* ] *command-string*

## AVAILABILITY

This command is available with the *uucp* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**uux** will gather 0 or more files from various systems, execute a command on a specified system and send the standard output to a file on a specified system.

For security reasons, most installations limit the list of commands executable on behalf of an incoming request from **uux**, permitting only the receipt of mail (see **mail**(1)). Remote execution permissions are defined in **/etc/uucp/Permissions**.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by '*system-name*!'. A null *system-name* is interpreted as the local system.

Path names may be one of:

- a full pathname;
- a pathname preceded by ~*username*/; *username* is interpreted as a username on the specified system and is replaced by that user's login directory on that system;
- a pathname preceded by ~/*destination*/; the ~/ is replaced by the "public UUCP" directory on the remote machine Note: this destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example ~/dan/ as the destination will make the directory **/usr/spool/uucppublic/dan** if it does not exist and put the requested file(s) in that directory;
- a partial pathname, which is prefixed by the pathname of current directory.

The '–' option sends the standard input to the **uux** command as the standard input to the *command-string*.

Any special shell characters such as **<>**, **;**, and **|** should be quoted either by quoting the entire *command-string*, or by quoting the special characters as individual arguments.

**uux** will attempt to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

**uux a!cut –f1 b!/usr/file \(c!/usr/file\)**

gets **/usr/file** from system **b** and sends it to system **a**, performs a **cut** command on that file and sends the result of the **cut** command to system **c**.

**uux** will notify you if the requested command on the remote system was disallowed, or if the command fails (that is, returns a non-zero exit status). This notification can be turned off by the **–n** option. The response comes by remote mail from the remote machine.

## OPTIONS

**–b** Return whatever standard input was provided to the **uux** command if the job fails (that is, returns a non-zero exit status).

**–c** Do not copy local file to the spool directory for transfer to the remote machine. This is the default.

**–C** Force the copy of local files to the spool directory for transfer.

**–j** Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by **uustat**(1C) to obtain the status or terminate a job.

**-n** Do not return any indication by **mail**(1) of success or failure of the job.

**-p** Same as '-': the standard input to **uux** is made the standard input to the *command-string*.

**-r** Do not start **uucico**(8C), just queue the job.

**-z** Return an indication by **mail** even if the job succeeds (that is, returns a zero exit status).

**-a***name* Use *name* as the user identification replacing the initiator user ID. Notification will be returned to the user.

**-g***grade*

*grade* is a single letter or number, from **0** to **9**, **A** to **Z**, or **a** to **z**; **0** is the highest grade, and **z** is the lowest grade. Lower grades will transmit the job earlier during a particular conversation. The default *grade* is **A**.

**-x** *debug_level*

Produce debugging output on the standard output. *debug_level* is a number between 0 and 9; higher numbers give more detailed information. 5, 7, and 9 are good numbers to try; they give increasing amounts of detail.

## EXAMPLE

The command

**uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"**

will get the **file1** and **file2** files from the **usg** and **pwba** machines, execute a **diff**(1) command and put the results in **file.diff** in the local **PUBDIR/dan** directory.

## FILES

| | |
|---|---|
| **/var/spool/uucp** | spool directories |
| **/etc/uucp/Permissions** | remote execution permissions |
| **/etc/uucp/*** | other data |
| **/usr/lib/uucp/*** | other programs |

## SEE ALSO

**mail**(1), **uucp**(1C), **uustat**(1C), **uucico**(8C)

*System and Network Administration*

## WARNINGS

Only the first command of a shell pipeline may have a '*system-name*!'. All other commands are executed on the system of the first command.

The use of the shell metacharacter * will probably not do what you want it to do.

The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the UUCP system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the **uux** request. The following command will NOT work:

**uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"**

but the command

**uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"**

will work (if **diff** is a permitted command).

When invoking **uux** from **csh**(1), the ! character must be prefixed by the \ escape to inhibit **csh**'s history mechanism. Quotes are not sufficient.

**BUGS**

Protected files and files that are in protected directories that are owned by the requestor can be sent in commands using **uux**. However, if the requestor is root, and the directory is not searchable by "other", the request will fail.

## NAME

vacation – reply to mail automatically

## SYNOPSIS

**vacation** [ **–I** ]
**vacation** [ **–j** ] [ **–a** *alias* ] [ **–t***N* ] *username*

## DESCRIPTION

**vacation** automatically replies to incoming mail. The reply is contained in the file **.vacation.msg** in your home directory. The **vacation** program run interactively will create a **.vacation.msg** file for you (which you may edit). Type **vacation** with no arguments. (See **USAGE** below.)

For example, the message created by **vacation** is:

**Subject: away from my mail**
**From: smith (via the vacation program)**
**I will not be reading my mail for a while.**
**Your mail regarding "$SUBJECT" will be read when I return**

The **.vacation.msg** file should include a header with at least a '**Subject:**' line (it should not contain a '**To:**' line and need not contain a '**From:**' line, since these are generated automatically).

If the string **$SUBJECT** appears in the **.vacation.msg** file, it is replaced with the subject of the original message when the reply is sent.

No message is sent if the '**To:**' or the '**Cc:**' line does not list the user to whom the original message was sent or one of a number of aliases for them, if the initial **From** line includes the string **–REQUEST@**, or if a '**Precedence: bulk**' or '**Precedence: junk**' line is included in the header.

## OPTIONS

**–I** Initialize the **.vacation.pag** and **.vacation.dir** files and start **vacation**.

If the **–I** flag is not specified, and a *user* argument is given, **vacation** reads the first line from the standard input (for a '**From:**' line, no colon). If absent, it produces an error message. The following options may be specified:

**–a** *alias* Indicate that *alias* is one of the valid aliases for the user running **vacation**, so that mail addressed to that alias generates a reply.

**–j** Do not check whether the recipient appears in the '**To:**' or the '**Cc:**' line.

**–t***N* Change the interval between repeat replies to the same sender. The default is 1 week. A trailing **s**, **m**, **h**, **d**, or **w** scales *N* to seconds, minutes, hours, days, or weeks respectively.

## USAGE

To start **vacation**, create a **.forward** file in your home directory containing a line of the form:

*username*, "|/usr/ucb/vacation *username*"

where *username* is your login name.

Then type in the command:

**vacation –I**

To stop **vacation**, remove the **.forward** file, or move it to a new name.

If **vacation** is run with no arguments, it will permit you to interactively turn **vacation** on or off. It will create a **.vacation.msg** file for you, or edit an existing one, using the editor specified by the **VISUAL** or **EDITOR** environment variable, or **vi**(1) if neither of those environment variables are set. If a **.forward** file is present in your home directory, it will ask whether you want to remove it and turn off **vacation**. If it is not present in your home directory, it creates it for you, and automatically performs a '**vacation –I**' function, turning on **vacation**.

**FILES**

**.forward**
**$HOME/.vacation.msg**

A list of senders is kept in the files **.vacation.pag** and **.vacation.dir** in your home directory.

**SEE ALSO**

**vi**(1), **sendmail**(8)

**NAME**

vfontinfo – inspect and print out information about fonts

**SYNOPSIS**

**/usr/lib/vfontinfo** [ **–mvz** ] *fontname* [ *characters* ]

**AVAILABILITY**

This command is available with the *Versatec* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**vfontinfo** allows you to examine a font in the UNIX system format. It prints out all the information in the font header and information about every non-null (width > 0) glyph. This can be used to make sure the font is consistent with the format.

The *fontname* argument is the name of the font you wish to inspect. It writes to the standard output. If it cannot find the file in your working directory, it looks in **/usr/lib/vfont** (the place most of the fonts are kept).

The *characters*, if given, specify certain characters to show. If omitted, the entire font is shown.

**OPTIONS**

**–m** Display header.

**–v** Verbose. Display bits of the glyph using commas, periods, and pipe ('|') symbols.

**–z** Zoom. Display bits of the glyph as an array of M and SPACE characters.

**FILES**

**/usr/lib/vfont**

**/usr/lib/vpd** Versatec daemon

**SEE ALSO**

**vswap**(1), **vwidth**(1), **vfont**(5)

## NAME

vgrind – grind nice program listings

## SYNOPSIS

**vgrind** [ **–fntwWx** ] [ **–d** *defs-file* ] [ **–h** *header* ] [ **–l***language* ] [ **–sn** ] [ **–o***pagelist* ] [ **–P***printer* ] [ **–T***output-device* ] *filename*...

## AVAILABILITY

This command is available with the *Text* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**vgrind** formats the program sources named by the *filename* arguments in a nice style using **troff**(1). Comments are placed in italics, keywords in bold face, and as each function is encountered its name is listed on the page margin.

**vgrind** runs in two basic modes, filter mode or regular mode. In filter mode **vgrind** acts as a filter in a manner similar to **tbl**(1). The standard input is passed directly to the standard output except for lines bracketed by the **troff**-like macros:

**.vS** starts processing

**.vE** ends processing

These lines are formatted as described above. The output from this filter can be passed to **troff** for output. There need be no particular ordering with **eqn**(1) or **tbl**.

In regular mode **vgrind** accepts input *filename*s, processes them, and passes them to **troff** for output. If no *filename* is given, or if the '–' argument is given, **vgrind** reads from the standard input (default if **–f** is specified).

In both modes **vgrind** passes any lines beginning with a decimal point without conversion.

## OPTIONS

Note: the syntax of options with arguments is important. Some require a SPACE between the option name and the argument, while those that do not have a SPACE below will not tolerate one.

**–f** Force filter mode.

**–n** Do not make keywords boldface.

**–x** Output the index file in a "pretty" format. The index file itself is produced whenever **vgrind** is run with a file called **index** present in the current directory. The index of function definitions can then be run off by giving **vgrind** the **–x** option and the file **index** as argument.

**–w** Consider TAB characters to be spaced four columns apart instead of the usual eight.

**–d** *defs-file*
Specify an alternate language definitions file (default is **/usr/lib/vgrindefs**).

**–h** *header*
Specify a header to appear in the center of every output page.

**–l***language*
Specify the language to use. Among the languages currently known are: Bourne shell (**–lsh**), C (**–lc**, the default), C shell (**–lcsh**), emacs MLisp, (**–lml**), FORTRAN (**–lf**), Icon (**–li**), ISP (**–i**), LDL (**–lLDL**), Model (**–lm**), Pascal (**–lp**), and RATFOR (**–lr**).

**–s***n* Specify a point size to use on output (exactly the same as the argument of a **troff .ps** point size request).

**vgrind** passes the following options to the formatter specified by the **TROFF** environment variable, see ENVIRONMENT below.

**–t** Similar to the same option in **troff**; that is, formatted text goes to the standard output.

**–o***pagelist*
Print only those pages whose page numbers appear in the comma-separated *pagelist* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end.

**–P***printer*
Send output to the named *printer*.

**–T***output-device*
Format output for the specified *output-device*.

**–W** Force output to the (wide) Versatec printer rather than the (narrow) Varian. **vtroff**(1) only.

## ENVIRONMENT

In regular mode **vgrind** feeds its intermediate output to the text formatter given by the value of the **TROFF** environment variable, or to **troff** if this variable is not defined in the environment. This mechanism allows for local variations in **troff**'s name.

## FILES

| | |
|---|---|
| **index** | file where source for index is created |
| **/usr/lib/vgrindefs** | language descriptions |
| **/usr/lib/vfontedpr** | preprocessor |
| **/usr/share/lib/tmac/tmac.vgrind** | macro package |

## SEE ALSO

**troff**(1), **vtroff**(1), **vgrindefs**(5)

## BUGS

**vgrind** assumes that a certain programming style is followed:

C Function names can be preceded on a line only by SPACE, TAB, or an asterisk. The parenthesized arguments must also be on the same line.

FORTRAN Function names need to appear on the same line as the keywords *function* or *subroutine*.

MLisp Function names should not appear on the same line as the preceding *defun*.

Model Function names need to appear on the same line as the keywords *is beginproc*.

Pascal Function names need to appear on the same line as the keywords *function* or *procedure*.

If these conventions are not followed, the indexing and marginal function name comment mechanisms will fail.

More generally, arbitrary formatting styles for programs mostly look bad. The use of SPACE characters to align source code fails miserably; if you plan to **vgrind** your program you should use TAB characters. This is somewhat inevitable since the fonts **vgrind** uses are variable width.

The mechanism of **ctags**(1) in recognizing functions should be used here.

The **–w** option is a crock, but there is no other way to achieve the desired effect.

The macros defined in **tmac.vgrind** do not coexist gracefully with those of other macro packages, making filter mode difficult to use effectively.

**vgrind** does not process certain special characters in **csh**(1) scripts correctly.

## NAME

vi, view, vedit – visual display editor based on ex(1)

## SYNOPSIS

**vi** [ **–CILRVx** ] [ **–c** *command* ] [ **–r** *filename* ] [ **–t** *tag* ] [ **–w***nnn* ] [ **+***command* ] *filename*...

**view**...

**vedit**...

## DESCRIPTION

**vi** (visual) is a display oriented text editor based on **ex**(1). **ex** and **vi** are, in fact, the same text editor; it is possible to get to the command mode of **ex** from within **vi** and vice-versa.

**view** runs **vi** with the **readonly** flag set. With **view**, you can browse through files interactively without making any changes.

**vedit** runs **vi** with the **report** flag set to 1, the **showmode** and **novice** flags set, and the **magic** flag turned off. These default settings are intended to make easier for beginners to learn **vi**.

## OPTIONS

**–C** Encryption option; the same as the **–x** option, except that all input text is assumed to have already been encrypted. This guarantees decryption in the cases where the **–x** option incorrectly determines that the input file is not already encrypted (this is extremely rare, and will only occur in conjunction with the use of files containing non-ASCII text).

**–l** Set up for editing LISP programs.

**–L** List the names of all files saved as the result of an editor or system crash.

**–R** Edit files in read only state. This has the same effect as the **view** command.

**–V** Verbose. Any non-tty input will be echoed on standard error.

**–x** Prompt for a key to be used in encrypting the file being edited. When used in conjunction with a pre-existing file, *ex* will make an educated guess to determine whether or not the input text file is already encrypted.

**–c** *command* Start the editing session by executing the editor command *command*. If *command* contains spaces, it must be surrounded by double quotess, see EXAMPLES below.

**–r** *filename* Recover the named files after a crash.

**–t** *tag* Edit the file containing *tag*. There must be a tags database in the directory, built by **ctags**(1), that contains a reference to *tag*.

**+***command* Start the editing session by executing *command*. This is identical to the **–c** option.

## ENVIRONMENT

The editor recognizes the environment variable **EXINIT** as a command (or list of commands separated by | characters) to run when it starts up. If this variable is undefined, the editor checks for startup commands in the file ~**/.exrc** file, which you must own. However, if there is a **.exrc** owned by you in the current directory, the editor takes its startup commands from this file — overriding both the file in your home directory and the environment variable.

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **vi**. On entry to **vi**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale. In the "C" locale, all 8-bit characters are escaped into an octal representation.

**EXAMPLES**

The following command:

**example% vi –c ":r test" tested**

will read in the file **test** at the end of the **tested** file.

**SEE ALSO**

**ctags**(1), **ex**(1)

*Editing Text Files*
*SunOS User's Guide: Getting Started*

**BUGS**

Software TAB characters using CTRL-T work only immediately after the **autoindent**.

SHIFT-left and SHIFT-right on intelligent terminals do not make use of insert and delete character operations in the terminal.

The **wrapmargin** option can be fooled since it looks at output columns when blanks are typed. When insert mode pushes an existing word through the margin and onto the next line without a break, the line will not be broken.

Insert/delete within a line can be slow if TAB characters are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as '**:source**'; there is no way to use the '**:append**', '**:change**', and '**:insert**' commands, since it is not possible to give more than one line of input to a '**:**' escape. To use these on a '**:global**' you must Q to *ex* command mode, execute them, and then reenter the screen editor with **vi** or **open**.

When using the **–r** option to recover a file, you must write the recovered text before quitting or you will lose it. **vi** does not prevent you from exiting without writing unless you make changes.

**vi** does not adjust when the SunView window in which it runs is resized.

**RESTRICTIONS**

The encryption facilities of **vi** are not available on software shipped outside the U.S.

**NAME**

vplot – plot graphics for a Versatec printer

**SYNOPSIS**

**vplot** [ **–VW** ] [ **–b** *lpr-argument* ] *filename*

**AVAILABILITY**

This command is available with the *Versatec* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**vplot** reads **plot**(5) format graphics input from the file specified by *filename* (the standard input if no *filename* is specified) and produces a plot on the Varian or Versatec printer.

**OPTIONS**

**–V** Force output to the standard Versatec printer.

**–W** Force output to the (wide) Versatec printer rather than the standard Versatec printer.

**–b** *lpr-argument*

*argument* (the next argument on the command line) specifies extra arguments to **lpr**(1).

**SEE ALSO**

**lpr**(1), **plot**(1G), **plot**(5)

## NAME

vswap – convert a foreign font file

## SYNOPSIS

**/usr/lib/vswap** [ **–r** ]

## AVAILABILITY

This command is available with the *Versatec* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

Without the **-r** option, **vswap** translates its standard input (which must be a **vfont**(5) file in the reversed-byte order) into a locally correct *vfont* file on its standard output. With the **-r** option, **vswap** translates its standard input (which must be a **vfont** file in the local-byte order) into a byte-reversed **vfont** file on its standard output.

The UNIX system **vfont** representation for fonts is a binary file containing machine-dependent elements — short (16-bit) integers, in particular. There are (at least) two common ways of representing a 16-bit integer. A program compiled on a VAX will expect the VAX format, while the same program compiled on a machine using a Motorola 68000-family processor or a SPARC processor will expect the reverse format. **vswap** can be used to convert font files created on a VAX to the format required to use them on Suns. (All Suns use the same **vfont** format, regardless of the native byte order of the processor, including Suns that use the Intel 80386 processor. Programs that will be run on Suns that use the Intel 80386 processor must be aware of this, and convert the machine-dependent elements when they read or write **vfont** files.) It can also convert Sun-format font files to VAX format (with the **–r** option).

## SEE ALSO

**troff**(1), **vfont**(5)

## BUGS

A machine-independent font format should be defined.

## NAME

vtroff – troff to a raster plotter

## SYNOPSIS

**vtroff** [ **–wx** ] [ **–F** *majorfont* ] [ **–l***length* ] [ **–123** *minorfont* ] *troff-arguments*

## AVAILABILITY

This command is available with the *Versatec* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**vtroff** runs **troff**(1) sending its output through various programs to produce typeset output on a raster plotter such as a Benson-Varian or a Versatec.

## OPTIONS

**–w** Specify that a wide output device be used; the default is to use a narrow device.

**–x** Simulate photo-typesetter output exactly. As with, using the width tables for the C.A.T. photo-typesetter.

**–F** *fontname*

Specify *fontname* as the desired font. This will place normal, italic and bold versions of the font on positions 1, 2, and 3. The default font is a Hershey font. argument and then the font name.

**–l***length*

Split the output onto successive pages every *length* inches rather than the default 11 inches.

**–123** *minorfont*

Place a font only on a single position specified by *–n* (where *n* is **1**, **2**, or **3**) and the minor font name. A **.r** will be added to the minor font name if needed. Thus

**vtroff –ms paper**

will set a paper in the Hershey font, while

**vtroff –F nonie –ms paper**

will set the paper in the (sans serif) nonie font.

## FILES

| | |
|---|---|
| **/usr/share/lib/tmac/tmac.vcat** | default font mounts and bug fixes |
| **/usr/lib/fontinfo/*** | fixes for other fonts |
| **/usr/lib/vfont** | directory containing fonts |

## SEE ALSO

**troff**(1), **vfont**(5)

## BUGS

Since some macro packages work correctly only if the fonts named R, I, B, and S are mounted, and since the Versatec fonts have different widths for individual characters than the fonts found on the typesetter, the following dodge was necessary: If you do not use the **.fp troff** directive then you get the widths of the standard typesetter fonts suitable for shipping the output of troff over the network to the computer center A machine for phototypesetting. If, however, you remount the R, I, B and S fonts, then you get the width tables for the Versatec.

**NAME**

vwidth – make a troff width table for a font

**SYNOPSIS**

**/usr/lib/vwidth** *fontfile pointsize* **> ftxx.c**

**cc -c ftxx.c**

**mv ftxx.o /usr/lib/font/ftxx**

**AVAILABILITY**

This command is available with the *Versatec* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**vwidth** translates from the width information stored in the **vfont** style format to the format expected by **troff**(1) — an object file in **a.out**(5) format. **troff** should look directly in the font file but it doesn't.

**vwidth** should be used after editing a font with **fontedit**(1). It is not necessary to use **vwidth** unless you have made a change that would affect the width tables. Such changes include numerically editing the width field, adding a new character, and moving or copying a character to a new position. It is *not* always necessary to use **vwidth** if the physical width of the glyph (for instance the number of columns in the bit matrix) has changed, but if it has changed much the logical width should probably be changed and **vwidth** run.

**vwidth** produces a C program on its standard output. This program should be run through the C compiler and the object (that is, the **.o** file) saved. The resulting file should be placed in **/usr/lib/font** in the file ft*xx* where *xx* is a one or two letter code that is the logical (internal to **troff**) font name. This name can be found by looking in the file **/usr/lib/fontinfo/***fname** where *fname* is the external name of the font.

**FILES**

**/usr/lib/font**
**a.out**

**SEE ALSO**

**troff**(1), **vtroff**(1), **fontedit**(1), **a.out**(5), **vfont**(5)

## NAME

w – who is logged in, and what are they doing

## SYNOPSIS

**w** [ **–hls** ] [ *user* ]

## DESCRIPTION

w displays a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields displayed are: the users login name, the name of the tty the user is on, the time of day the user logged on (in hours:minutes), the idle time — that is, the number of minutes since the user last typed anything (in hours:minutes), the CPU time used by all processes and their children on that terminal (in minutes:seconds), the CPU time used by the currently active processes (in minutes:seconds), the name and arguments of the current process.

If a *user* name is included, output is restricted to that user.

## OPTIONS

**–h** Suppress the heading.

**–l** Produce a long form of output, which is the default.

**–s** Produce a short form of output. In the short form, the tty is abbreviated, the login time and CPU times are left off, as are the arguments to commands.

## EXAMPLE

```
example% w
7:36am  up 6 days, 16:45,  1 users,  load average: 0.20, 0.23, 0.18
User     tty      login@  idle   JCPU   PCPU  what
ralph    console  7:10am     1   10:05   4:31  w
example%
```

## ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout w. On entry to w, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## FILES

**/etc/utmp**
**/dev/kmem**
**/dev/drum**

## SEE ALSO

**ps**(1), **who**(1), **utmp**(5V)

## BUGS

The notion of the "current process" is muddy. The current algorithm is 'the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal'. This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. In cases where no process can be found, w prints '–'.

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is "charged" with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

w does not know about the new conventions for detecting background jobs. It will sometimes find a background job instead of the right one.

**NAME**

wait – wait for a process to finish

**SYNOPSIS**

**wait**

**DESCRIPTION**

Wait until all processes started with **&** or **bg** have completed, and report on abnormal terminations.

Because the **wait**(2V) system call must be executed in the parent process, the shell itself executes **wait**, without creating a new process.

**SEE ALSO**

**csh**(1), **sh**(1), **wait**(2V)

**BUGS**

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for. This bug does not apply to **csh**(1).

**NAME**

wall – write to all users logged in

**SYNOPSIS**

**wall** [ **–a** ] [ *filename* ]

**DESCRIPTION**

**wall** reads the standard input until an EOF. It then sends this message, preceded by '**Broadcast Message ...**', to all logged in users. If *filename* is given, then the message is read in from that file. Normally, pseudo-terminals that do not correspond to rlogin sessions are ignored. Thus when in **sunview**(1), the message appears only on the console window. However, **–a** will send the message even to such pseudo-terminals.

The sender should be super-user to override any protections the users may have invoked.

**FILES**

**/dev/tty?**
**/etc/utmp**

**SEE ALSO**

**mesg**(1), **sunview**(1), **write**(1)

**NAME**

wc – display a count of lines, words and characters

**SYNOPSIS**

**wc** [ **–lwc** ] [ *filename* ... ]

**DESCRIPTION**

**wc** counts lines, words, and characters in *filename*s, or in the standard input if no *filename* appears. It also keeps a total count for all named files. A word is a string of characters delimited by SPACE, TAB, or NEW-LINE characters.

**OPTIONS**

When *filename*s are specified on the command line, their names will be printed along with the counts.

The default is **–lwc** (count lines, words, and characters).

**l** Count lines.

**w** Count words.

**c** Count characters.

**EXAMPLE**

```
example%
wc /usr/share/man/man1/{csh.1,sh.1,telnet.1}
    1876   11223   65895  /usr/share/man/man1/csh.1
     674    3310   20338  /usr/share/man/man1/sh.1
     260    1110    6834  /usr/share/man/man1/telnet.1
    2810   15643   93067  total
example%
```

**ENVIRONMENT**

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **wc**. On entry to **wc**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

## NAME

what – extract SCCS version information from a file

## SYNOPSIS

**what** [ **–s** ] *filename* ...

## DESCRIPTION

**what** searches each *filename* for occurrences of the pattern **@(#)** that the SCCS **get** command (see **sccs-get**(1)) substitutes for the **%Z%** ID keyword, and prints what follows up to a ", >, NEWLINE, \, or null character. For instance, if the C program in file **program.c** contains

**char sccsid[ ] = " @(#)identification information ";**

and **program.c** is compiled to yield **program.o** and **a.out**, the command:

**what program.c program.o a.out**

produces:

**program.c:**
identification information

**program.o:**
identification information

**a.out:** identification information

## OPTIONS

**–s** Stop after the first occurrence of the pattern.

## SEE ALSO

**sccs**(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **sccs-val**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

## DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

## BUGS

There is a remote possibility that a spurious occurrence of the '@(#)' pattern could be found by **what.**

## NAME

whatis – display a one-line summary about a keyword

## SYNOPSIS

**whatis** *command*...

## DESCRIPTION

**whatis** looks up a given *command* and displays the header line from the manual section. You can then run the **man**(1) command to get more information. If the line starts '**name**(*section*)...' you can do '**man** *section* **name**' to get the documentation for it. Try '**whatis ed**' and then you should do '**man 1 ed**' to get the manual page for **ed**(1).

**whatis** is actually just the **–f** option to the **man**(1) command.

**whatis** uses the **whatis** database. This database can be created using the **–w** option of **catman**(8).

## FILES

**/usr/[share]/man/whatis**
data base

## SEE ALSO

**apropos**(1), **man**(1), **catman**(8)

## NOTES

**whatis** uses the **/usr/man/whatis** database, which is created by **catman**(8).

## NAME

whereis – locate the binary, source, and manual page files for a command

## SYNOPSIS

**whereis** [ **–bmsu** ] [ **–BMS** *directory*... **–f** ] *filename* ...

## DESCRIPTION

**whereis** locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form *.ext*, for example, **.c**. Prefixes of **s.** resulting from use of source code control are also dealt with. **whereis** then attempts to locate the desired program in a list of standard places:

**/usr/bin**
**/usr/bin**
**/usr/5bin**
**/usr/games**
**/usr/hosts**
**/usr/include**
**/usr/local**
**/usr/etc**
**/usr/lib**
**/usr/share/man**
**/usr/src**
**/usr/ucb**

## OPTIONS

**–b** Search only for binaries.

**–m** Search only for manual sections.

**–s** Search only for sources.

**–u** Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus '**whereis –m –u ***' asks for those files in the current directory which have no documentation.

**–B** Change or otherwise limit the places where **whereis** searches for binaries.

**–M** Change or otherwise limit the places where **whereis** searches for manual sections.

**–S** Change or otherwise limit the places where **whereis** searches for sources.

**–f** Terminate the last directory list and signals the start of file names, and *must* be used when any of the **–B**, **–M**, or **–S** options are used.

## EXAMPLE

Find all files in **/usr/bin** which are not documented in **/usr/share/man/man1** with source in **/usr/src/cmd**:

```
example% cd /usr/ucb
example% whereis –u –M /usr/share/man/man1 –S /usr/src/cmd –f *
```

## FILES

**/usr/src/***
**/usr/{doc,man}/***
**/etc, /usr/{lib,bin,ucb,old,new,local}**

## SEE ALSO

**chdir**(2V)

**BUGS**

Since **whereis** uses **chdir**(2V) to run faster, pathnames given with the **–M**, **–S**, or **–B** must be full; that is, they must begin with a '/'.

**NAME**

which – locate a command; display its pathname or alias

**SYNOPSIS**

**which** [ *filename* ] ...

**DESCRIPTION**

**which** takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's **.cshrc** file.

**FILES**

| | |
|---|---|
| ~/**.cshrc** | source of aliases and path values |

**SEE ALSO**

**csh**(1)

**DIAGNOSTICS**

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

**BUGS**

Only aliases and paths from ~/**.cshrc** are used; importing from the current environment is not attempted. Must be executed by **csh**(1), since only **csh** knows about aliases.

To compensate for ~/**.cshrc** files in which aliases depend upon the **prompt** variable being set, **which** sets this variable. If the ~/**.cshrc** produces output or prompts for input when **prompt** is set, **which** may produce some strange results.

## NAME

who – who is logged in on the system

## SYNOPSIS

**who** [ **who-file** ] [ **am i** ]

## DESCRIPTION

Used without arguments, **who** lists the login name, terminal name, and login time for each current user. **who** gets this information from the **/etc/utmp** file.

If a filename argument is given, the named file is examined instead of **/etc/utmp**. Typically the named file is **/var/adm/wtmp**, which contains a record of all logins since it was created. In this case, **who** lists logins, logouts, and crashes. Each login is listed with user name, terminal name (with **/dev/** suppressed), and date and time. Logouts produce a similar line without a user name. Reboots produce a line with '~' in place of the device name, and a fossil time indicating when the system went down. Finally, the adjacent pair of entries '|' and '}' indicate the system-maintained time just before and after a **date** command changed the system's idea of the time.

With two arguments, as in '**who am i**' (and also '**who is who**'), **who** tells who you are logged in as: it displays your hostname, login name, terminal name, and login time.

## EXAMPLES

```
example% who am i
example!ralph   ttyp0    Apr 27 11:24
example%

example% who
mktg      ttym0  Apr 27 11:11
gwen      ttyp0  Apr 27 11:25
ralph     ttyp1  Apr 27 11:30
example%
```

## FILES

**/etc/utmp**
**/var/adm/wtmp**

## SEE ALSO

**login**(1), w(1), **whoami**(1), **utmp**(5V), **locale**(5)

**NAME**

whoami – display the effective current username

**SYNOPSIS**

**whoami**

**DESCRIPTION**

**whoami** displays the login name corresponding to the current effective uer ID. If you have used **su**(1V) to temporarily adopt another user, **whoami** will report the login name associated with that user ID. **whoami** gets its information from the **geteuid()** and **getpwuid()** library routines (see **getuid**(2V) and **getpwent**(3V), respectively).

**FILES**

| | |
|---|---|
| **/etc/passwd** | username data base |
| **/etc/utmp** | database of users currently logged in |

**SEE ALSO**

**su**(1V), **who**(1), **getuid**(2V), **getpwent**(3V), **utmp**(5V)

**NAME**

whois – TCP/IP Internet user name directory service

**SYNOPSIS**

**whois** [ **–h** *host* ] *identifier*

**DESCRIPTION**

**whois** searches for an TCP/IP directory entry for an *identifier* which is either a name (such as "Smith") or a handle (such as "SRI-NIC"). You can force a name-only search by preceding the name with a period; you can force a handle-only search by preceding the handle with an exclamation point. See EXAMPLES.

If you are searching for a group or organization entry, you can have the entire membership list of the group displayed with the record by preceding the argument with '*' (an asterisk).

You may of course use an exclamation point and asterisk, or a period and asterisk together.

**EXAMPLES**

**example% whois Smith**

looks for name or handle SMITH.

**example% whois \!SRI-NIC**

looks for handle SRI-NIC only.

**example% whois '.Smith, John'**

looks for name "John Smith" only.

Adding '...' to the name or handle argument will match anything from that point; that is, 'ZU...' will match ZUL, ZUM, etc.

**NAME**

write – write a message to another user

**SYNOPSIS**

**write** *username* [ *ttyname* ]

**DESCRIPTION**

**write** copies lines from your standard input to *username*'s screen.

When you type a **write** command, the person you are writing to sees a message like this:

**Message from** *hostname*!*yourname* **on** *yourttyname*

After typing the **write** command, enter the text of your message. What you type appears line-by-line on the other user's screen. Conclude by typing an EOF indication (CTRL-D) or an interrupt. At this point **write** displays EOF on your recipient's screen and exits.

To write to a user who is logged in more than once, use the *ttyname* argument to indicate the appropriate terminal name.

You can grant or deny other users permission to write to you by using the **mesg** command (default allows writing). Certain commands, **nroff**(1) and **pr**(1V) in particular, do not allow anyone to write to you while you are using them in order to prevent messy output.

If **write** finds the character '**!**' at the beginning of a line, it calls the shell to execute the rest of the line as a command.

Two people can carry on a conversation by "writing" to each other. When the other person receives the message indicating you are writing to him, he can then **write** back to you if he wishes. However, since you are now simultaneously typing and receiving messages, you end up with garbage on your screen unless you work out some sort of scheduling scheme with your partner. You might try the following conventional protocol: when you first write to another user, wait for him to write back before starting to send. Each person should end each message with a distinctive signal — **-o-** (for "over") is standard — so that the other knows when to begin a reply. To end your conversation, type **-oo-** (for "over and out") before finishing the conversation.

**EXAMPLE**

Here is an example of a short dialog between two people on different terminals. Two users called "Horace" and "Eudora" are logged in on a system called "jones". To illustrate the process, both users' screens are shown side-by-side:

| Eudora's Terminal | Horace's Terminal |
|---|---|
| | *Horace is staring at his screen* |
| **jones% write horace** | |
| **how about a squash game tonight? -o-** | |
| | **Message from jones!eudora on tty09 at 17:05 ...** |
| | **how about a squash game tonight? -o-** |
| | **jones% write eudora** |
| | **I'm playing tiddlywinks with Carmeline -o-** |
| **Message from jones!horace on tty03 at 17:06 ...** | |
| **I'm playing tiddlywinks with Carmeline -o-** | |
| **How about the beach on Sunday? -o-** | |
| | **How about the beach on Sunday? -o-** |
| | **Sorry, I'm washing my tent that day -o-** |
| **Sorry, I'm washing my tent that day -o-** | |
| **See you when I get back from Peru -oo-** | |

```
                                    See you when I get back from Peru -oo-
^D
jones%                              EOF
                                    I hear rack of llama is very tasty -oo-
                                    ^D
I hear rack of llama is very tasty -oo-
EOF                                 jones%
```

**ENVIRONMENT**

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **write**. On entry to **write**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 "C" locale.

**FILES**

| | |
|---|---|
| **/etc/utmp** | to find user |
| **/usr/bin/sh** | to execute ! |

**SEE ALSO**

**mail**(1), **mesg**(1), **pr**(1V), **talk**(1), **troff**(1), **who**(1), **locale**(5)

## NAME

xargs – construct the arguments list(s) and execute a command

## SYNOPSIS

**xargs** [ **–ptx** ] [ **–e***eofstr* ] [ **–i***replstr* ] [ **–l***number* ] [ **–n***number* ] [ **–s***size* ]
[ *command* [ *initial-arguments* ] ]

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**xargs** combines the fixed *initial-arguments* with arguments read from the standard input, to execute the specified *command* one or more times. The number of arguments read for each *command* invocation, and the manner in which they are combined are determined by the options specified.

*command*, which may be a shell file, is searched for using one's **$PATH**. If *command* is omitted, **/usr/bin/echo** is used.

Arguments read in from the standard input are defined to be contiguous strings of characters delimited by white space. Empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if they are escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings, a '\' (backslash) will escape the character it precedes.

Each arguments-list is constructed starting with the *initial-arguments*, followed by some number of arguments read from the standard input (Exception: see the **–i** option). Options **–i**, **–l**, and **–n** determine how arguments are selected for each command invocation. When none of these options are coded, the *initial-arguments* are followed by arguments read continuously from the standard input until an internal buffer is full, and then *command* is executed with the accumulated arguments. This process is repeated until there are none left. When there are option conflicts (for instance, **–l** versus **–n**), the last option takes precedence.

**xargs** will terminate if it receives a return code of **–1**, or if it cannot execute *command*. When *command* is a shell script, it should explicitly **exit** (see **sh**(1)) with an appropriate value to avoid accidentally returning with **–1**.

## OPTIONS

**–p** Prompt mode. The user is asked whether to execute *command* each invocation. Trace mode (**–t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a return, skips that particular invocation of *command*.

**–t** Trace mode. The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

**–x** Terminate **xargs** if any argument list would be greater than *size* characters; **–x** is forced by the options **–i** and **–l**. When neither of the options **–i**, **–l**, or **–n** are coded, the total length of all arguments must be within the *size* limit.

**–e***eofstr* *eofstr* is taken as the logical EOF string. '_' (underbar) is assumed for the logical **EOF** string if **–e** is not coded. The value **–e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). **xargs** reads the standard input until either EOF or the logical **EOF** string is encountered.

**–i***replstr* Insert mode: *command* is executed for each line from the standard input, taking the entire line as a single argument, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more

instances of *replstr*. SPACE and TAB characters at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **–x** is also forced. {} is assumed for *replstr* if not specified.

**–l***number* *command* is executed for each nonempty *number* lines of arguments from the standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first NEWLINE *unless* the last character of the line is a SPACE or a TAB; a trailing SPACE or TAB signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. The **–x** option is forced.

**–n***number* Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **–x** is also coded, each *number* arguments must fit in the *size* limitation, else **xargs** terminates execution.

**–s***size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **–s** is not coded, 470 is taken as the default. Note: the character count for *size* includes one extra character for each argument and the count of characters in the command name.

## EXAMPLES

The following will move all files from directory **$1** to directory **$2**, and echo each move command just before doing it:

```
ls $1 | xargs –i –t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file **log**:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into **arch**(1) one at a time, or many at a time.

```
ls | xargs –p –l ar r arch
ls | xargs –p –l | xargs ar r arch
```

The following will execute **diff**(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs –n2 diff
```

## SEE ALSO

**arch**(1), **diff**(1), **sh**(1)

**NAME**

xsend, xget, enroll – send or receive secret mail

**SYNOPSIS**

**xsend** *username*

**xget**

**enroll**

**DESCRIPTION**

These commands implement a secure communication channel, which is like **mail**(1), but no one can read the messages except the intended recipient. The method embodies a public-key cryptosystem using knapsacks.

To receive messages, use **enroll**; it asks you for a password that you must subsequently quote in order to receive secret mail.

To receive secret mail, use **xget**. It asks for your password, then gives you the messages.

To send secret mail, use **xsend** in the same manner as the ordinary mail command. Unlike **mail**, **xsend** accepts only one target. A message announcing the receipt of secret mail is also sent by ordinary mail.

**FILES**

**/var/spool/secretmail/*.key** keys
**/var/spool/secretmail/*.[0-9]** messages

**SEE ALSO**

**mail**(1)

**BUGS**

The knapsack public-key cryptosystem is known to be breakable.

Secret mail should be integrated with ordinary mail.

The announcement of secret mail makes "traffic analysis" possible.

## NAME

xstr – extract strings from C programs to implement shared strings

## SYNOPSIS

**xstr** **–c** *filename* [ **–v** ] [ **–l** *array* ]
**xstr** [ **–l** *array* ]
**xstr** *filename* [ **–v** ] [ **–l** *array* ]

## DESCRIPTION

**xstr** maintains a file called **strings** into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, which are most useful if they are also read-only.

The command

**xstr** **–c** *filename*

extracts the strings from the C source in name, replacing string references by expressions of the form **&xstr**[*number*] for some number. An appropriate declaration of **xstr** is prepended to the file. The resulting C text is placed in the file **x.c**, to then be compiled. The strings from this file are placed in the **strings** data base if they are not there already. Repeated strings and strings which are suffixes of existing strings do not cause changes to the data base.

After all components of a large program have been compiled, a file declaring the common **xstr** space called **xs.c** can be created by a command of the form

**xstr**

This **xs.c** file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

**xstr** can also be used on a single file. A command

**xstr** *filename*

creates files **x.c** and **xs.c** as before, without using or affecting any **strings** file in the same directory.

It may be useful to run **xstr** after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. **xstr** reads from the standard input when the argument '–' is given. An appropriate command sequence for running **xstr** after the C preprocessor is:

```
cc –E name.c | xstr –c –
cc –c x.c
mv x.o name.o
```

**xstr** does not touch the file **strings** unless new items are added; thus **make**(1) can avoid remaking **xs.o** unless truly necessary.

## OPTIONS

| | |
|---|---|
| **–c** *filename* | Take C source text from *filename*. |
| **–v** | Verbose: display a progress report indicating where new or duplicate strings were found. |
| **–l** *array* | Specify the named *array* in program references to abstracted strings. The default array name is **xstr**. |

**FILES**

| | |
|---|---|
| **strings** | data base of strings |
| **x.c** | massaged C source |
| **xs.c** | C source for definition of array "xstr" |
| **/tmp/xs*** | temp file when **xstr** *filename* doesn't touch **strings** |

**SEE ALSO**

**make**(1), **mkstr**(1)

**BUGS**

If a string is a suffix of another string in the data base, but the shorter string is seen first by **xstr** both strings will be placed in the data base, when just placing the longer one there would do.

## NAME

yacc – yet another compiler-compiler: parsing program generator

## SYNOPSIS

**yacc** [ **–dvlt** ] *grammar*

## DESCRIPTION

**yacc** converts a context–free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous, in which case specified precedence rules may be used to break those ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a function named **yyparse**(). The **yyparse**() function must be loaded with the lexical analyzer **yylex**(), as well as **main**() and the error handling routine **yyerror**(). These routines must be supplied by the user; **lex**(1) is useful for creating lexical analyzers usable by **yacc**–produced parsers.

## OPTIONS

**–d** Generate the file **y.tab.h** with the **define** statements that associate the **yacc**–assigned "token codes" with the user–declared "token names" so that source files other than **y.tab.c** can access the token codes.

**–v** Prepare the file **y.output** containing a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

**–l** Generate **y.tab.c** containing no numbered line directives.

**–t** Set the preprocessor symbol **yydebug**, so **y.tab.c** will be compiled with runtime debugging code.

## FILES

| | |
|---|---|
| **y.output** | description of parsing tables and conflict report |
| **y.tab.c** | output parser |
| **y.tab.h** | defines for token names |
| **yacc.tmp, yacc.acts** | temporary files |
| **/usr/lib/yaccpar** | parser prototype for C programs |
| **/usr/5lib/liby.a** | library provides primitive routines **main**() and **yyerror**() to facilitate the initial use of **yacc**. |

## SEE ALSO

**cc**(1V), **lex**(1)

*Programming Utilities and Libraries*
*LR Parsing* by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974

## DIAGNOSTICS

The number of reduce–reduce and shift–reduce conflicts is reported on the standard output; a more detailed report is found in the **y.output** file. Similarly, it is also reported if some rules are not reachable from the start symbol.

## NOTES

Because the **cc**(1V) command does not generate or support 8-bit symbol names, it is inappropriate to make **yacc** 8-bit clean. See **cc**(1V) for an explanation about why **cc** is not 8-bit clean.

## BUGS

Because file names are fixed, no more than one **yacc** process should be active in a given directory at a time.

**NAME**

yes – be repetitively affirmative

**SYNOPSIS**

**yes** [ *expletive* ]

**DESCRIPTION**

**yes** repeatedly outputs **y**, or if *expletive* is given, that is output repeatedly. Termination is by typing an interrupt character.

**NAME**

ypcat – print values in a NIS data base

**SYNOPSIS**

**ypcat** [ **–kt** ] [ **–d** *domainname* ] *mname*

**ypcat –x**

**AVAILABILITY**

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**ypcat** prints out values in a Network Information Service (NIS) map specified by *mname*, which may be either a map name or a map nickname . Since **ypcat** uses the NIS service, no NIS server is specified.

To look at the network-wide password database, **passwd.byname**, (with the nickname **passwd**). type in:

**ypcat passwd**

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the NIS service.

**OPTIONS**

**–k** Display the keys for those maps in which the values are null or the key is not part of the value.

**–t** Inhibit translation of *mname* to map name. For example, '**ypcat –t passwd**' will fail because there is no map named **passwd**, whereas ' **ypcat passwd** ' will be translated to 'ypcat passwd.byname'.

**–d** *domainname*
Specify a domain other than the default domain. The default domain is returned by *domainname*.

**–x** Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

**SEE ALSO**

**domainname**(1), **ypmatch**(1), **ypfiles**(5), **ypserv**(8)

**NOTES**

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

**NAME**

ypmatch – print the value of one or more keys from a NIS map

**SYNOPSIS**

**ypmatch** [ **–d** *domain* ] [ **–k** ] [ **–t** ] *key* ... *mname*

**ypmatch –x**

**AVAILABILITY**

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

**DESCRIPTION**

**ypmatch** prints the values associated with one or more keys from the Network Information Service (NIS) map specified by *mname*, which may be either a *mapname* or an map nickname.

Multiple keys can be specified; the same map will be searched for all . The keys must be exact values insofar as capitalization and length are concerned. No pattern matching is available. If a key is not matched, a diagnostic message is produced.

**OPTIONS**

**–d** Specify a domain other than the default domain.

**–k** Before printing the value of a key, print the key itself, followed by a '**:**' colon . This is useful only if the keys are not duplicated in the values, or you've specified so many keys that the output could be confusing.

**–t** Inhibit translation of nickname to *mapname*. For example, '**ypmatch –t zippy passwd** ' will fail because there is no map named **passwd**, while '**ypmatch zippy passwd** ' will be translated to '**ypmatch zippy passwd.byname** '.

**–x** Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

**SEE ALSO**

**ypcat**(1), **ypfiles**(5)

**NOTES**

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

## NAME

yppasswd – change your network password in the NIS database

## SYNOPSIS

**yppasswd** [ *username* ]

**ypchfn** [ *username* ]

**ypchsh** [ *username* ]

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**yppasswd** changes (or installs) the network password associated with the user *username* (your own name by default) in the Network Information Service (NIS) database. The NIS password may be different from the local one on your own machine. See **passwd**(1).

**ypchfn** changes the full name associated with *username* in the NIS database.

**ypchsh** changes the login shell associated with *username* in the NIS database.

**yppasswd** prompts for the old NIS password, and then for the new one. You must type in the old password correctly for the change to take effect. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long, if they use a sufficiently rich alphabet, and at least six characters long if monocase. These rules are relaxed if you are insistent enough. Only the owner of the name or the super-user may change a password; in either case you must prove you know the old password.

The NIS password daemon, **yppasswdd**(8C) must be running on your NIS server in order for the new password to take effect.

## SEE ALSO

**passwd**(1), **ypfiles**(5), **yppasswdd**(8C)

## BUGS

The update protocol passes all the information to the server in one RPC call, without ever looking at it. Thus if you type in your old password incorrectly, you will not be notified until after you have entered your new password.

## NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

## NAME

ypwhich – return hostname of NIS server or map master

## SYNOPSIS

**ypwhich** [ **–d** [ *domain* ] ] [ **–V1** | **–V2** ] [ *hostname* ]
**ypwhich** [ **–t** *mapname* ] [ **–d** *domain* ] **–m** [ *mname* ]
**ypwhich –x**

## AVAILABILITY

This command is available with the *Networking* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

## DESCRIPTION

**ypwhich** tells which Network Information Service (NIS) server supplies NIS services to an NIS client, or which is the master for a map. If invoked without arguments, it gives the NIS server for the local machine. If *hostname* is specified, that machine is queried to find out which NIS server it is using.

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the NIS services.

## OPTIONS

**–d** Use *domain* instead of the default domain.

**–V1** Which server is serving v.1 NIS protocol-speaking client processes.

**–V2** Which server is serving v.2 NIS protocol client processes.

If neither version is specified, **ypwhich** attempts to locate the server that supplies the (current) v.2 services. If there is no v.2 server currently bound, **ypwhich** then attempts to locate the server supplying the v.1 services. Since NIS servers and NIS clients are both backward compatible, the user need seldom be concerned about which version is currently in use.

**–t** *mapname*

Inhibit nickname translation; useful if there is a *mapname* identical to a nickname. This is not true of any Sun-supplied map.

**–m** Find the master NIS server for a map. No *hostname* can be specified with the **–m** option. *mname* can be a mapname, or a nickname for a map. When *mname* is omitted, produce a list available maps.

**–x** Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

## SEE ALSO

**ypfiles**(5), **rpcinfo**(8C), **ypserv**(8), **ypset**(8)

## NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

# Notes