

**USER'S MANUAL**  
**RELEASE 5.0**

**UNISYS\***  
**system**



**Western Electric**

# **UNIX System User's Manual**

**Release 5.0**

**June 1982**

*Not for use or disclosure outside the  
Bell System except under written agreement.*

**UNIX is a trademark of Bell Laboratories**

Copyright © 1982 Bell Laboratories, Inc.

*This manual was set on an AUTOLOGIC, Inc.  
APS-5 phototypesetter driven by the TROFF  
formatter operating under the UNIX system.*

## INTRODUCTION

This manual describes the features of UNIX. It provides neither a general overview of UNIX (for that, see "The UNIX Time-Sharing System," *BSTJ*, Vol. 57, No. 6, Part 2, pp. 1905-29, by D. M. Ritchie and K. Thompson), nor details of the implementation of the system (see "UNIX Implementation," *BSTJ*, same issue, pp. 1931-46).

Not all commands, features, and facilities described in this manual are available in every UNIX system. The entries not applicable for a particular hardware line will have an appropriate caveat stamped in the center of the mast of an entry. Also, programs or facilities being phased out will be marked as "Obsolescent" on the top of the entry. When in doubt, consult your system's administrator.

This manual is divided into six sections, some containing inter-filed sub-classes:

1. Commands and Application Programs:
  1. General-Purpose Commands.
  - 1C. Communications Commands.
  - 1G. Graphics Commands.
2. System Calls.
3. Subroutines:
  - 3C. C and Assembler Library Routines.
  - 3F. FORTRAN Library Routines.
  - 3M. Mathematical Library Routines.
  - 3S. Standard I/O Library Routines.
  - 3X. Miscellaneous Routines.
4. File Formats.
5. Miscellaneous Facilities.
6. Games.

**Section 1** (*Commands and Application Programs*) describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory `/bin` (for binary programs). Some programs also reside in `/usr/bin`, to save space in `/bin`. These directories are searched automatically by the command interpreter called the *shell*. Sub-class 1C contains communication programs such as *cu*, *send*, *uucp*, etc. These entries may not apply from system to system depending upon the hardware included on your processor. Some UNIX systems may have a directory called `/usr/lbin`, containing local commands.

**Section 2** (*System Calls*) describes the entries into the UNIX kernel, including the C language interface.

**Section 3** (*Subroutines*) describes the available subroutines. Their binary versions reside in various system libraries in the directories `/lib` and `/usr/lib`. See *intro*(3) for descriptions of these libraries and the files in which they are stored.

**Section 4** (*File Formats*) documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(4). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language `struct` declarations corresponding to these formats can be found in the directories `/usr/include` and `/usr/include/sys`.

**Section 5** (*Miscellaneous Facilities*) contains a variety of things. Included are descriptions of character sets, macro packages, etc.

**Section 6** (*Games*) describes the games and educational programs that, as a rule, reside in the directory `/usr/games`.

## Introduction

Each section consists of a number of independent entries of a page or so each. The name of the entry appears in the upper corners of its pages. Entries within each section are alphabetized, with the exception of the introductory entry that begins each section. The page numbers of each entry start at 1. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

All entries are based on a common format, not all of whose parts always appear:

The **NAME** part gives the name(s) of the entry and briefly states its purpose.

The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (*Commands*):

**Boldface** strings are literals and are to be typed just as they appear.

*Italic* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual (they are underlined in the typed version of the entries).

Square brackets [] around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file", it always refers to a *file* name.

Ellipses ... are used to show that the previous argument prototype may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus -, plus +, or equal sign = is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with -, +, or =.

The **DESCRIPTION** part discusses the subject at hand.

The **EXAMPLE(S)** part gives example(s) of usage, where appropriate.

The **FILES** part gives the file names that are built into the program.

The **SEE ALSO** part gives pointers to related information.

The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The **WARNINGS** part points out potential pitfalls.

The **BUGS** part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents and a permuted index derived from that table precede Section 1. On each *index* line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

On most systems, all entries are available on-line via the *man*(1) command, q.v.

## HOW TO GET STARTED

This discussion provides the basic information you need to get started on UNIX: how to log in and log out, how to communicate through your terminal, and how to run a program. (See the *UNIX User's Guide* for a more complete introduction to the system.)

**Logging in.** You must dial up UNIX from an appropriate terminal. UNIX supports full-duplex ASCII terminals. You must also have a valid user name, which may be obtained (together with the telephone number(s) of your UNIX system) from the administrator of your system. Common terminal speeds are 10, 15, 30, and 120 characters per second (110, 150, 300, and 1,200 baud); occasionally, speeds of 240, 480, and 960 characters per second (2,400, 4,800, and 9,600 baud) are also available. On some UNIX systems, there are separate telephone numbers for each available terminal speed, while on other systems several speeds may be served by a single telephone number. In the latter case, there is one "preferred" speed; if you dial in from a terminal set to a different speed, you will be greeted by a string of meaningless characters (the **login:** message at the wrong speed). Keep hitting the "break" or "attention" key until the **login:** message appears. Hard-wired terminals usually are set to the correct speed.

Most terminals have a speed switch that should be set to the appropriate speed and a half-/full-duplex switch that should be set to full-duplex. When a connection (at the speed of the terminal) has been established, the system types **login:** and you then type your user name followed by the "return" key. If you have a password (and you should!), the system asks for it, but does not print ("echo") it on the terminal. After you have logged in, the "return", "new-line", and "line-feed" keys will give exactly the same result.

It is important that you type your login name in lower case if possible; if you type upper-case letters, UNIX will assume that your terminal cannot generate lower-case letters and that you mean all subsequent upper-case input to be treated as lower case. When you have logged in successfully, the shell will type a \$ to you. (The shell is described below under *How to run a program.*)

For more information, consult *login(1)*, which discuss the login sequence in more detail, and *stty(1)*, which tells you how to describe the characteristics of your terminal to the system (*profile(4)* explains how to accomplish this last task automatically every time you log in).

**Logging out.** There are two ways to log out:

1. You can simply hang up the phone.
2. You can log out by typing an end-of-file indication (ASCII EOT character, usually typed as "control-d") to the shell. The shell will terminate and the **login:** message will appear again.

**How to communicate through your terminal.** When you type to UNIX, a gnome deep in the system is gathering your characters and saving them. These characters will not be given to a program until you type a "return" (or "new-line"), as described above in *Logging in.*

UNIX terminal input/output is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is typing at you. Of course, if you type during output, the output will have interspersed in it the input characters. However, whatever you type will be saved and interpreted in the correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system throws away *all* the saved characters.

On an input line from a terminal, the character @ “kills” all the characters typed before it. The character # erases the last character typed. Successive uses of # will erase characters back to, but not beyond, the beginning of the line; @ and # can be typed as themselves by preceding them with \ (thus, to erase a \, you need two #s). These default erase and kill characters can be changed; see *stty(1)*.

The ASCII DC3 (control-s) character can be used to temporarily stop output. It is useful with CRT terminals to prevent output from disappearing before it can be read. Output is resumed when a DC1 (control-q) or a second DC3 (or any other character, for that matter) is typed. The DC1 and DC3 characters are not passed to any other program when used in this manner.

The ASCII DEL (a.k.a. “rubout”) character is not passed to programs, but instead generates an *interrupt signal*, just like the “break”, “interrupt”, or “attention” signal. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you don’t want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor *ed(1)*, for example, catches interrupts and stops what it is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited.

The *quit* signal is generated by typing the ASCII FS character. It not only causes a running program to terminate, but also generates a file with the “core image” of the terminated process. *Quit* is useful for debugging.

Besides adapting to the speed of the terminal, UNIX tries to be intelligent as to whether you have a terminal with the “new-line” function, or whether it must be simulated with a “carriage-return” and “line-feed” pair. In the latter case, all *input* “carriage-return” characters are changed to “line-feed” characters (the standard line delimiter), and a “carriage-return” and “line-feed” pair is echoed to the terminal. If you get into the wrong mode, the *stty(1)* command will rescue you.

Tab characters are used freely in UNIX source programs. If your terminal does not have the tab function, you can arrange to have tab characters changed into spaces during output, and echoed as spaces during input. Again, the *stty(1)* command will set or reset this mode. The system assumes that tabs are set every eight character positions. The *tabs(1)* command will set tab stops on your terminal, if that is possible.

**How to run a program.** When you have successfully logged into UNIX, a program called the shell is listening to your terminal. The shell reads the lines you type, splits them into a command name and its arguments, and executes the command. A command is simply an executable program. Normally, the shell looks first in your current directory (see *The current directory* below) for a program with the given name, and if none is there, then in system directories. There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and arrange for the shell to find them there.

The command name is the first word on an input line to the shell; the command and its arguments are separated from one another by space and/or tab characters.

When a program terminates, the shell will ordinarily regain control and type a \$ at you to indicate that it is ready for another command. The shell has many other capabilities, which are described in detail in *sh(1)*.

**The current directory.** UNIX has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he or she also created a directory for you (ordinarily with the same name as your user name, and known as your *login* or *home* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name you type is by default assumed

to be in that directory. Because you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. Permissions to have your will with other directories and files will have been granted or denied to you by their respective owners, or by the system administrator. To change the current directory use *cd(1)*.

**Path names.** To refer to files not in the current directory, you must use a path name. Full path names begin with /, which is the name of the *root* directory of the whole file system. After the slash comes the name of each directory containing the next sub-directory (followed by a /), until finally the file name is reached (e.g., /usr/ae/filex refers to file *filex* in directory *ae*, while *ae* is itself a subdirectory of *usr*; *usr* springs directly from the root directory). See *intro(2)* for a formal definition of *path name*.

If your current directory contains subdirectories, the path names of files therein begin with the name of the corresponding subdirectory (*without* a prefixed /). Without important exception, a path name may be used anywhere a file name is required.

Important commands that modify the contents of files are *cp(1)*, *mv*, and *rm(1)*, which respectively copy, move (i.e., rename), and remove files. To find out the status of files or directories, use *ls(1)*. Use *mkdir(1)* for making directories and *rmdir(1)* for destroying them.

For a fuller discussion of the file system, see the references cited at the beginning of the *INTRODUCTION* above. It may also be useful to glance through Section 2 of this manual, which discusses system calls, even if you don't intend to deal with the system at that level.

**Writing a program.** To enter the text of a source program into a UNIX file, use *ed(1)*. The principal languages available under UNIX are C (see *cc(1)*), Fortran (see *f77(1)*), and assembly language (see *as(1)*). After the program text has been entered with the editor and written into a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. Normally, the output of the language processor will be left in a file in the current directory named *a.out* (if that output is precious, use *mv(1)* to give it a less vulnerable name). If the program is written in assembly language, you will probably need to load with it library subroutines (see *ld(1)*). Fortran and C call the loader automatically.

When you have finally gone through this entire process without provoking any diagnostics, the resulting program can be run by giving its name to the shell in response to the \$ prompt.

If any execution (run-time) errors occur, you will need *sdb(1)* to examine the remains of your program.

Your programs can receive arguments from the command line just as system programs do; see *exec(2)*.

**Text processing.** Almost all text is entered through the editor *ed(1)*. The commands most often used to write text on a terminal are *cat(1)*, *pr(1)*, and *nroff*. The *cat(1)* command simply dumps ASCII text on the terminal, with no processing at all. The *pr(1)* command paginates the text, supplies headings, and has a facility for multi-column output. *Nroff* is an elaborate text formatting program, and requires careful forethought in entering both the text and the formatting commands into the input file; it produces output on a typewriter-like terminal. *Troff(1)* is very similar to *nroff*, but produces its output on a phototypesetter (it was used to typeset this manual). There are several "macro" packages (especially the so-called *mm* package) that significantly ease the effort required to use *nroff* and *troff(1)*; Section 5 entries for these packages indicate where you can find their detailed



descriptions.

**Surprises.** Certain commands provide *inter-user* communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may aim them at you. To communicate with another user currently logged in, *write(1)* is used; *mail(1)* will leave a message whose presence will be announced to another user when he or she next logs in. The corresponding entries in this manual also suggest how to respond to these two commands if you are their target.

When you log in, a message-of-the-day may greet you before the first \$.

## TABLE OF CONTENTS

### 1. Commands and Application Programs

intro	introduction to commands and application programs
300	handle special functions of DASI 300 and 300s terminals
4014	paginator for the Tektronix 4014 terminal
450	handle special functions of the DASI 450 terminal
acctcom	search and print process accounting file(s)
adb	absolute debugger
admin	create and administer SCCS files
ar	archive and library maintainer for portable archives
ar.pdp	archive and library maintainer
arcv	convert archive files from PDP-11 to common archive format
as	common assembler
as.pdp	assembler for PDP-11
asa	interpret ASA carriage control characters
awk	pattern scanning and processing language
banner	make posters
basename	deliver portions of path names
bc	arbitrary-precision arithmetic language
bdiff	big diff
bfs	big file scanner
bs	a compiler/interpreter for modest-sized programs
cal	print calendar
calendar	reminder service
cat	concatenate and print files
cb	C program beautifier
cc	C compiler
cd	change working directory
cdc	change the delta commentary of an SCCS delta
cflow	generate C flow graph
chmod	change mode
chown	change owner or group
cmp	compare two files
col	filter reverse line-feeds
comb	combine SCCS deltas
comm	select or reject lines common to two sorted files
convert	convert object and archive files to common formats
cp	copy, link or move files
cpio	copy file archives in and out
cpp	the C language preprocessor
cprs	compress an IS25 object file
crypt	encode/decode
csplit	context split
ct	spawn getty to a remote terminal
cu	call another UNIX system
cut	cut out selected fields of each line of a file
cw	prepare constant-width text for troff
cxref	generate C program cross reference
date	print and set the date
dc	desk calculator
dd	convert and copy a file
delta	make a delta (change) to an SCCS file
deroff	remove nroff/troff, tbl, and eqn constructs
diff	differential file comparator
diff3	3-way differential file comparison

diffmk . . . . . mark differences between files

dircmp . . . . . directory comparison

dis . . . . . 3B20S disassembler

dpd . . . . . HONEYWELL sending daemon, line printer daemon

dpr . . . . . off-line print

du . . . . . summarize disk usage

dump . . . . . dump selected parts of an object file

echo . . . . . echo arguments

ed . . . . . text editor

efl . . . . . Extended Fortran Language

enable . . . . . enable/disable LP printers

env . . . . . set environment for command execution

eqn . . . . . format mathematical text for nroff or troff

expr . . . . . evaluate arguments as an expression

f77 . . . . . Fortran 77 compiler

factor . . . . . factor a number

fget . . . . . retrieve files from the HONEYWELL 6000

file . . . . . determine file type

find . . . . . find files

fsend . . . . . send files to the HONEYWELL 6000

fsplit . . . . . split f77, ratfor, or efl files

gcat . . . . . send phototypesetter output to the HONEYWELL 6000

gcosmail . . . . . send mail to HIS user

gdev . . . . . graphical device routines and filters

ged . . . . . graphical editor

get . . . . . get a version of an SCCS file

getopt . . . . . parse command options

graph . . . . . draw a graph

graphics . . . . . access graphical and numerical commands

greek . . . . . select terminal filter

grep . . . . . search a file for a pattern

gutil . . . . . graphical utilities

help . . . . . ask for help

hp . . . . . handle special functions of HP 2640 and 2621-series terminals

hpio . . . . . HP 2645A terminal tape file archiver

hyphen . . . . . find hyphenated words

id . . . . . print user and group IDs and names

ipcrm . . . . . remove a message queue, semaphore set or shared memory id

ipcs . . . . . report inter-process communication facilities status

join . . . . . relational database operator

kasb . . . . . assembler/un-assembler for the KMC11B microprocessor

kill . . . . . terminate a process

ld . . . . . link editor for common object files

ld.pdp . . . . . link editor

lex . . . . . generate programs for simple lexical tasks

line . . . . . read one line

lint . . . . . a C program checker

list . . . . . produce C source listing from 3B20S object file

login . . . . . sign on

logname . . . . . get login name

lorder . . . . . find ordering relation for an object library

lp . . . . . send/cancel requests to an LP line printer

lpr . . . . . line printer spooler

lpstat . . . . . print LP status information

ls . . . . . list contents of directories

m4 . . . . . macro processor

machid . . . . . provide truth value about your processor type

mail	send mail to users or read mail
make	maintain, update, and regenerate groups of programs
makekey	generate encryption key
man	print entries in this manual
mesg	permit or deny messages
mkdir	make a directory
mm	print/check documents formatted with the MM macros
mmt	typeset documents, view graphs, and slides
net	execute a command on the PCL network
newform	change the format of a text file
newgrp	log in to a new group
news	print news items
nice	run a command at low priority
nl	line numbering filter
nm	print name list of common object file
nm.pdp	print name list
nohup	run a command immune to hangups and quits
nroff	format text
nscstat	query the operation status of the NSC network
nscctorje	re-route jobs from the NSC network to RJE
nusend	send files to another UNIX on the NSC network
od	octal dump
pack	compress and expand files
passwd	change login password
paste	merge same lines of several files or subsequent lines of one file
pr	print files
prof	display profile data
prs	print an SCCS file
ps	report process status
ptx	permuted index
pwd	working directory name
ratfor	rational Fortran dialect
regcmp	regular expression compile
rjstat	RJE status report and interactive status console
rm	remove files or directories
rmdel	remove a delta from an SCCS file
sact	print current SCCS file editing activity
sadp	disk access profiler
sag	system activity graph
sar	system activity reporter
scat	concatenate and print files on synchronous printer
scc	C compiler for stand-alone programs
sccsdiff	compare two versions of an SCCS file
sdb	symbolic debugger
sdiff	side-by-side difference program
se	screen editor for video terminals
sed	stream editor
send	gather files and/or submit RJE jobs
sh	shell, the standard/restricted command programming language
size	print section sizes of common object files
size.pdp	print sizes of object files
sleep	suspend execution for an interval
sno	SNOBOL interpreter
sort	sort and/or merge files
spell	find spelling errors
spline	interpolate smooth curve
split	split a file into pieces

stat	statistical network useful with graphical commands
stlogin	sign on to synchronous terminal
strip	strip symbol and line number information from a common object file
strip.pdp	remove symbols and relocation bits
ststat	report synchronous terminal facilities status
stty	set the options for a terminal
su	become super-user or another user
sum	print checksum and block count of a file
sync	update the super block
tabs	set tabs on a terminal
tail	deliver the last part of a file
tar	tape file archiver
tbl	format tables for nroff or troff
tc	phototypesetter simulator
tee	pipe fitting
test	condition evaluation command
time	time a command
timex	time a command; report process data and system activity
toc	graphical table of contents routines
touch	update access and modification times of a file
tplot	graphics filters
tr	translate characters
troff	typeset text
trouble	log a trouble report
true	provide truth values
tsort	topological sort
tty	get the terminal's name
umask	set file-creation mode mask
uname	print name of current UNIX system
unget	undo a previous get of an SCCS file
uniq	report repeated lines in a file
units	conversion program
uucp	unix to unix copy
uustat	uucp status inquiry and job control
uuto	public UNIX-to-UNIX file copy
uux	unix to unix command execution
val	validate SCCS file
vc	version control
vpr	Versatec printer spooler
wait	await completion of process
wc	word count
what	identify SCCS files
who	who is on the system
write	write to another user
xargs	construct argument list(s) and execute command
yacc	yet another compiler-compiler

**2. System Calls**

intro	introduction to system calls and error numbers
access	determine accessibility of a file
acct	enable or disable process accounting
alarm	set a process's alarm clock
brk	change data segment space allocation
chdir	change working directory
chmod	change mode of file
chown	change owner and group of a file

chroot	change root directory
close	close a file descriptor
creat	create a new file or rewrite an existing one
dup	duplicate an open file descriptor
exec	execute a file
exit	terminate process
fcntl	file control
fork	create a new process
getpid	get process, process group, and parent process IDs
getuid	get real user, effective user, real group, and effective group IDs
ioctl	control device
kill	send a signal to a process or a group of processes
link	link to a file
lseek	move read/write file pointer
maus	multiple-access-user-space (shared memory) operations
mknod	make a directory, or a special or ordinary file
mount	mount a file system
msgctl	message control operations
msgget	get message queue
msgop	message operations
nice	change priority of a process
open	open for reading or writing
pause	suspend process until signal
pipe	create an interprocess channel
plock	lock process, text, or data in memory
profil	execution time profile
ptrace	process trace
read	read from file
semctl	semaphore control operations
semget	get set of semaphores
semop	semaphore operations
setpgrp	set process group ID
setuid	set user and group IDs
shmctl	shared memory control operations
shmget	get shared memory segment
shmop	shared memory operations
signal	specify what to do upon receipt of a signal
stat	get file status
stime	set time
sync	update super-block
sys3b	3B20S specific system calls
time	get time
times	get process and child process times
ulimit	get and set user limits
umask	set and get file creation mask
umount	unmount a file system
uname	get name of current UNIX system
unlink	remove directory entry
ustat	get file system statistics
utime	set file access and modification times
wait	wait for child process to stop or terminate
write	write on a file

### 3. Subroutines

intro	introduction to subroutines and libraries
a64l	convert between long integer and base-64 ASCII string

abort . . . . . generate an IOT fault  
 abort . . . . . terminate Fortran program  
 abs . . . . . return integer absolute value  
 abs . . . . . Fortran absolute value  
 acos . . . . . Fortran arccosine intrinsic function  
 aimag . . . . . Fortran imaginary part of complex argument  
 aint . . . . . Fortran integer part intrinsic function  
 asin . . . . . Fortran arcsine intrinsic function  
 assert . . . . . verify program assertion  
 atan . . . . . Fortran arctangent intrinsic function  
 atan2 . . . . . Fortran arctangent intrinsic function  
 atof . . . . . convert ASCII string to floating-point number  
 bessel . . . . . Bessel functions  
 bool . . . . . Fortran bitwise boolean functions  
 bsearch . . . . . binary search  
 clock . . . . . report CPU time used  
 conjg . . . . . Fortran complex conjugate intrinsic function  
 conv . . . . . translate characters  
 cos . . . . . Fortran cosine intrinsic function  
 cosh . . . . . Fortran hyperbolic cosine intrinsic function  
 crypt . . . . . generate DES encryption  
 ctermid . . . . . generate file name for terminal  
 ctime . . . . . convert date and time to string  
 ctype . . . . . classify characters  
 cuserid . . . . . get character login name of the user  
 dial . . . . . establish an out-going terminal line connection  
 drand48 . . . . . generate uniformly distributed pseudo-random numbers  
 ecvt . . . . . convert floating-point number to string  
 end . . . . . last locations in program  
 erf . . . . . error function and complementary error function  
 exp . . . . . Fortran exponential intrinsic function  
 exp . . . . . exponential, logarithm, power, square root functions  
 fclose . . . . . close or flush a stream  
 ferror . . . . . stream status inquiries  
 floor . . . . . floor, ceiling, remainder, absolute value functions  
 fopen . . . . . open a stream  
 fread . . . . . binary input/output  
 frexp . . . . . manipulate parts of floating-point numbers  
 fseek . . . . . reposition a file pointer in a stream  
 ftw . . . . . walk a file tree  
 ftype . . . . . explicit Fortran type conversion  
 gamma . . . . . log gamma function  
 getarg . . . . . return Fortran command-line argument  
 getc . . . . . get character or word from stream  
 getcwd . . . . . get path-name of current working directory  
 getenv . . . . . return value for environment name  
 getenv . . . . . return Fortran environment variable  
 getgrent . . . . . get group file entry  
 getlogin . . . . . get login name  
 getopt . . . . . get option letter from argument vector  
 getpass . . . . . read a password  
 getpw . . . . . get name from UID  
 getpwent . . . . . get password file entry  
 gets . . . . . get a string from a stream  
 getut . . . . . access utmp file entry  
 hsearch . . . . . manage hash search tables  
 hypot . . . . . Euclidean distance function

index	return location of Fortran substring
l3tol	convert between 3-byte integers and long integers
ldahread	read the archive header of a member of an archive file
ldclose	close a common object file
ldfhread	read the file header of a common object file
ldread	manipulate line number entries of a common object file function
ldseek	seek to line number entries of a section of a common object file
ldohseek	seek to the optional file header of a common object file
ldopen	open a common object file for reading
ldrseek	seek to relocation entries of a section of a common object file
ldshread	read an indexed/named section header of a common object file
ldsseek	seek to an indexed/named section of a common object file
ldtbindx	compute the index of a symbol table entry of a common object file
ldtbread	read an indexed symbol table entry of a common object file
ldtbseek	seek to the symbol table of a common object file
len	return length of Fortran string
log	Fortran natural logarithm intrinsic function
log10	Fortran common logarithm intrinsic function
logname	return login name of user
lsearch	linear search and update
malloc	main memory allocator
matherr	error-handling function
max	Fortran maximum-value functions
mclock	return Fortran time accounting
memory	memory operations
min	Fortran minimum-value functions
mktemp	make a unique file name
mod	Fortran remaindering intrinsic functions
monitor	prepare execution profile
nlist	get entries from name list
perror	system error messages
plot	graphics interface subroutines
popen	initiate pipe to/from a process
printf	print formatted output
putc	put character or word on a stream
putpwent	write password file entry
puts	put a string on a stream
qsort	quicker sort
rand	simple random-number generator
rand	Fortran uniform random-number generator
regcmp	compile and execute regular expression
round	Fortran nearest integer functions
scanf	convert formatted input
setbuf	assign buffering to a stream
setjmp	non-local goto
sign	Fortran transfer-of-sign intrinsic function
signal	specify Fortran action on receipt of a system signal
sin	Fortran sine intrinsic function
sinh	Fortran hyperbolic sine intrinsic function
sinh	hyperbolic functions
sleep	suspend execution for interval
sputl	access long numeric data in a machine independent fashion.
sqrt	Fortran square root intrinsic function
ssignal	software signals
stdio	standard buffered input/output package
stdipc	standard interprocess communication package
string	string operations



strtol . . . . . convert string to integer  
 swab . . . . . swap bytes  
 system . . . . . issue a shell command from Fortran  
 system . . . . . issue a shell command  
 tan . . . . . Fortran tangent intrinsic function  
 tanh . . . . . Fortran hyperbolic tangent intrinsic function  
 tmpfile . . . . . create a temporary file  
 tmpnam . . . . . create a name for a temporary file  
 trig . . . . . trigonometric functions  
 tsearch . . . . . manage binary search trees  
 ttyname . . . . . find name of a terminal  
 ttyslot . . . . . find the slot in the utmp file of the current user  
 ungetc . . . . . push character back into input stream  
 x25alnk . . . . . attach or install a BX.25 link  
 x25clnk . . . . . change over a BX.25 link  
 x25hlnk . . . . . halt or detach a BX.25 link  
 x25ipvc . . . . . install or remove a PVC on a link

**4. File Formats**

intro . . . . . introduction to file formats  
 a.out . . . . . common assembler and link editor output  
 a.out.pdp . . . . . PDP-11 assembler and link editor output  
 acct . . . . . per-process accounting file format  
 ar . . . . . common archive file format  
 ar.pdp . . . . . archive file format  
 checklist . . . . . list of file systems processed by fsck  
 core . . . . . format of core image file  
 cpio . . . . . format of cpio archive  
 dir . . . . . format of directories  
 errfile . . . . . error-log file format  
 filehdr . . . . . file header for common object files  
 fs . . . . . format of system volume  
 fspec . . . . . format specification in text files  
 gettydefs . . . . . speed and terminal settings used by getty  
 gps . . . . . graphical primitive string, format of graphical files  
 group . . . . . group file  
 inittab . . . . . script for the init process  
 inode . . . . . format of an inode  
 issue . . . . . issue identification file  
 ldfcn . . . . . common object file access routines  
 linenum . . . . . line number entries in a common object file  
 master.dec . . . . . master device information table  
 master.u3b . . . . . master device information table  
 mnttab . . . . . mounted file system table  
 passwd . . . . . password file  
 plot . . . . . graphics interface  
 pnch . . . . . file format for card images  
 profile . . . . . setting up an environment at login time  
 reloc . . . . . relocation information for a common object file  
 sccsfile . . . . . format of SCCS file  
 scnhdr . . . . . section header for a common object file  
 syms . . . . . common object file symbol table format  
 system . . . . . format of 3B20S system description file  
 utmp . . . . . utmp and wtmp entry formats

**5. Miscellaneous Facilities**

intro	introduction to miscellany
ascii	map of ASCII character set
environ	user environment
eqnchar	special character definitions for eqn and neqn
fcntl	file control options
greek	graphics for the extended TTY-37 type-box
man	macros for formatting entries in this manual
mm	the MM macro package for formatting documents
mosd	the OSDD adapter macro package for formatting documents
mptx	the macro package for formatting a permuted index
mv	a troff macro package for typesetting view graphs and slides
regexp	regular expression compile and match routines
stat	data returned by stat system call
term	conventional names for terminals
types	primitive system data types

**6. Games**

intro	introduction to games
arithmetic	provide drill in number facts
back	the game of backgammon
bj	the game of black jack
chess	the game of chess
craps	the game of craps
hangman	guess the word
jotto	secret word game
maze	generate a maze
moo	guessing game
quiz	test your knowledge
reversi	a game of dramatic reversals
sky	obtain ephemerides
ttt	tic-tac-toe
wump	the game of hunt-the-wumpus



## PERMUTED INDEX

<ul style="list-style-type: none"> <li>/functions of HP 2640 and</li> <li>handle special functions of HP archiver. hpio: HP</li> <li>functions of DASI 300 and/</li> <li>/special functions of DASI of DASI 300 and 300s/ 300, functions of DASI 300 and dis:</li> <li>produce C source listing from sys3b:</li> <li>    system: format of</li> <li>l3tol, ltol3: convert between comparison. diff3:</li> <li>    Tektronix 4014 terminal.</li> <li>    paginator for the Tektronix of the DASI 450 terminal.</li> <li>special functions of the DASI files from the HONEYWELL</li> <li>send files to the HONEYWELL output to the HONEYWELL</li> <li>    f77: Fortran</li> <li>    long integer and base-64/</li> <li>    program.</li> <li>    Fortran absolute value.</li> <li>    adb:</li> <li>    abs: return integer</li> <li>    dabs, cabs, zabs: Fortran</li> <li>    /floor, ceiling, remainder, of a file. touch: update</li> <li>    utime: set file</li> <li>    accessibility of a file.</li> <li>    commands. graphics:</li> <li>    machine/ sputl, sgetl:</li> <li>    sadb: disk</li> <li>    ldfcn: common object file</li> <li>/setutent, endutent, utmpname:</li> <li>    access: determine</li> <li>    enable or disable process</li> <li>    acct: per-process</li> <li>    search and print process</li> <li>    mclock: return Fortran time</li> <li>    process accounting.</li> <li>    file format.</li> <li>    process accounting file(s).</li> <li>    sin, cos, tan, asin,</li> <li>    intrinsic function.</li> <li>    sag: system</li> <li>    sar: system</li> <li>    current SCCS file editing</li> <li>report process data and system</li> <li>formatting/ mosd: the OSDD</li> <li>    SCCS files.</li> <li>    admin: create and</li> <li>imaginary part of complex/</li> <li>part intrinsic function.</li> <li>alarm: set a process's</li> <li>    clock.</li> <li>change data segment space</li> </ul>	<ul style="list-style-type: none"> <li>2621-series terminals. . . . . hp(1)</li> <li>2640 and 2621-series/ hp: . . . . . hp(1)</li> <li>2645A terminal tape file . . . . . hpio(1)</li> <li>300, 300s: handle special . . . . . 300(1)</li> <li>300 and 300s terminals. . . . . 300(1)</li> <li>300s: handle special functions . . . . . 300(1)</li> <li>300s terminals. /special . . . . . 300(1)</li> <li>3B20S disassembler. . . . . dis(1)</li> <li>3B20S object file. list: . . . . . list(1)</li> <li>3B20S specific system calls. . . . . sys3b(2)</li> <li>3B20S system description file. . . . . system(4)</li> <li>3-byte integers and long/ . . . . . l3tol(3C)</li> <li>3-way differential file . . . . . diff3(1)</li> <li>4014: paginator for the . . . . . 4014(1)</li> <li>4014 terminal. 4014: . . . . . 4014(1)</li> <li>450: handle special functions . . . . . 450(1)</li> <li>450 terminal. 450: handle . . . . . 450(1)</li> <li>6000. /fget.demon: retrieve . . . . . fget(1C)</li> <li>6000. fsend: . . . . . fsend(1C)</li> <li>6000. /send phototypesetter . . . . . gcat(1C)</li> <li>77 compiler. . . . . f77(1)</li> <li>a64l, l64a: convert between . . . . . a64l(3C)</li> <li>abort: generate an IOT fault. . . . . abort(3C)</li> <li>abort: terminate Fortran . . . . . abort(3F)</li> <li>abs, iabs, dabs, cabs, zabs: . . . . . abs(3F)</li> <li>abs: return integer absolute . . . . . abs(3C)</li> <li>absolute debugger. . . . . adb(1)</li> <li>absolute value. . . . . abs(3C)</li> <li>absolute value. abs, iabs, . . . . . abs(3F)</li> <li>absolute value functions. . . . . floor(3M)</li> <li>access and modification times . . . . . touch(1)</li> <li>access and modification times. . . . . utime(2)</li> <li>access: determine . . . . . access(2)</li> <li>access graphical and numerical . . . . . graphics(1G)</li> <li>access long numeric data in a . . . . . sputl(3X)</li> <li>access profiler. . . . . sadp(1)</li> <li>access routines. . . . . ldfcn(4)</li> <li>access utmp file entry. . . . . getut(3C)</li> <li>accessibility of a file. . . . . access(2)</li> <li>accounting. acct: . . . . . acct(2)</li> <li>accounting file format. . . . . acct(4)</li> <li>accounting file(s). acctcom: . . . . . acctcom(1)</li> <li>accounting. . . . . mclock(3F)</li> <li>acct: enable or disable . . . . . acct(2)</li> <li>acct: per-process accounting . . . . . acct(4)</li> <li>acctcom: search and print . . . . . acctcom(1)</li> <li>acos, atan, atan2:/ . . . . . trig(3M)</li> <li>acos, dacos: Fortran arccosine . . . . . acos(3F)</li> <li>activity graph. . . . . sag(1G)</li> <li>activity reporter. . . . . sar(1)</li> <li>activity. sact: print . . . . . sact(1)</li> <li>activity. /time a command; . . . . . timex(1)</li> <li>adapter macro package for . . . . . mosd(5)</li> <li>adb: absolute debugger. . . . . adb(1)</li> <li>admin: create and administer . . . . . admin(1)</li> <li>administer SCCS files. . . . . admin(1)</li> <li>aimag, dimag: Fortran . . . . . aimag(3F)</li> <li>aint, dint: Fortran integer . . . . . aint(3F)</li> <li>alarm clock. . . . . alarm(2)</li> <li>alarm: set a process's alarm . . . . . alarm(2)</li> <li>allocation. brk, sbrk: . . . . . brk(2)</li> </ul>
---	---

INDEX

INDEX

realloc, calloc: main memory allocator. malloc, free, . . . . . malloc(3C)  
     natural logarithm/ log, alog, dlog, clog: Fortran . . . . . log(3F)  
     logarithm intrinsic/ log10, alog10, dlog10: Fortran common . . . . . log10(3F)  
     Fortran/ max, max0, amax0, max1, dmax1: Fortran/ . . . . . max(3F)  
     max, max0, amax0, max1, amin0, min1, amin1, dmin1: . . . . . min(3F)  
     min, min0, amin0, min1, amin1, dmin1: Fortran/ . . . . . min(3F)  
 remaindering intrinsic/ mod, amod, dmod: Fortran . . . . . mod(3F)  
     rshift: Fortran bitwise/ and, or, xor, not, lshift, . . . . . bool(3F)  
         sort: sort and/or merge files. . . . . sort(1)  
         send, gath: gather files and/or submit RJE jobs. . . . . send(1C)  
     Fortran nearest integer/ anint, dnint, nint, idnint: . . . . . round(3F)  
         link editor output. a.out: common assembler and . . . . . a.out(4)  
         link editor output. a.out: PDP-11 assembler and . . . . . a.out.pdp(4)  
 introduction to commands and application programs. intro: . . . . . intro(1)  
     maintainer. ar: archive and library . . . . . ar.pdp(1)  
         maintainer for portable/ ar: archive and library . . . . . ar(1)  
             format. ar: archive file format. . . . . ar.pdp(4)  
             ar: common archive file . . . . . ar(4)  
             language. bc: arbitrary-precision arithmetic . . . . . bc(1)  
             acos, dacos: Fortran arccosine intrinsic function. . . . . acos(3F)  
             maintainer. ar: archive and library . . . . . ar.pdp(1)  
             for portable archives. ar: archive and library maintainer . . . . . ar(1)  
             cpio: format of cpio archive. . . . . cpio(4)  
             ar: common archive file format. . . . . ar(4)  
             archive file format. . . . . ar.pdp(4)  
             archive file. /the archive . . . . . ldahread(3X)  
             archive files from PDP-11 to . . . . . arcv(1)  
             convert: convert object and archive files to common/ . . . . . convert(1)  
             files from PDP-11 to common archive format. /archive . . . . . arcv(1)  
             an archive/ ldahread: read the archive header of a member of . . . . . ldahread(3X)  
             HP 2645A terminal tape file archiver. hpio: . . . . . hpio(1)  
             tar: tape file archiver. . . . . tar(1)  
             maintainer for portable archives. /archive and library . . . . . ar(1)  
             cpio: copy file archives in and out. . . . . cpio(1)  
             asin, dasin: Fortran arcsine intrinsic function. . . . . asin(3F)  
             atan2, datan2: Fortran arctangent intrinsic function. . . . . atan2(3F)  
             atan, datan: Fortran arctangent intrinsic function. . . . . atan(3F)  
 from PDP-11 to common archive/ arcv: convert archive files . . . . . arcv(1)  
     imaginary part of complex argument. /dimag: Fortran . . . . . aimag(3F)  
     return Fortran command-line argument. getarg: . . . . . getarg(3F)  
     command. xargs: construct argument list(s) and execute . . . . . xargs(1)  
     getopt: get option letter from argument vector. . . . . getopt(3C)  
         expr: evaluate arguments as an expression. . . . . expr(1)  
         echo: echo arguments. . . . . echo(1)  
         bc: arbitrary-precision arithmetic language. . . . . bc(1)  
         number facts. arithmetic: provide drill in . . . . . arithmetic(6)  
     expr: evaluate arguments as an expression. . . . . expr(1)  
         asa: assembler for PDP-11. . . . . as.pdp(1)  
         asa: common assembler. . . . . as(1)  
         ASA carriage control . . . . . asa(1)  
         asa: interpret ASA carriage . . . . . asa(1)  
         ASCII character set. ASCII character set. . . . . ascii(5)  
         ascii: map of ASCII character . . . . . ascii(5)  
         ASCII string. /convert between . . . . . a64i(3C)  
         ASCII string to floating-point . . . . . atof(3C)  
         asctime, tzset: convert date . . . . . ctime(3C)  
         asin, acos, atan, atan2: . . . . . trig(3M)  
         asin, dasin: Fortran arcsine . . . . . asin(3F)  
         ask for help. . . . . help(1)  
         output. a.out: common assembler and link editor . . . . . a.out(4)  
         output. a.out: PDP-11 assembler and link editor . . . . . a.out.pdp(4)  
             asa: common assembler. . . . . as(1)  
             asa: assembler for PDP-11. . . . . as.pdp(1)

KMC11B/ kasb, kunb:	assembler/un-assembler for the . . .	kasb(1)
assertion.	assert: verify program . . . . .	assert(3X)
assert: verify program	assertion. . . . .	assert(3X)
setbuf:	assign buffering to a stream. . . . .	setbuf(3S)
sin, cos, tan, asin, acos,	atan, atan2: trigonometric/ . . . . .	trig(3M)
arctangent intrinsic/	atan, datan: Fortran . . . . .	atan(3F)
arctangent intrinsic/	atan2, datan2: Fortran . . . . .	atan2(3F)
cos, tan, asin, acos, atan,	atan2: trigonometric/ sin, . . . . .	trig(3M)
floating-point number.	atof: convert ASCII string to . . . . .	atof(3C)
integer. strtol, atol,	atoi: convert string to . . . . .	strtol(3C)
integer. strtol,	atol, atoi: convert string to . . . . .	strtol(3C)
link. x25alnk, x25ilnk:	attach or install a BX.25 . . . . .	x25alnk(3C)
wait:	await completion of process. . . . .	wait(1)
processing language.	awk: pattern scanning and . . . . .	awk(1)
ungetc: push character	back into input stream. . . . .	ungetc(3S)
	back: the game of backgammon. . . . .	back(6)
	backgammon. . . . .	back(6)
	banner: make posters. . . . .	banner(1)
back: the game of	base-64 ASCII string. /convert . . . . .	a64i(3C)
between long integer and	basename, dirname: deliver . . . . .	basename(1)
portions of path names.	bc: arbitrary-precision . . . . .	bc(1)
arithmetic language.	bdiff: big diff. . . . .	bdiff(1)
	beautifier. . . . .	cb(1)
cb: C program	Bessel functions. . . . .	bessel(3M)
j0, j1, jn, y0, y1, yn:	bfs: big file scanner. . . . .	bfs(1)
	binary input/output. . . . .	fread(3S)
fread, fwrite:	binary search. . . . .	bsearch(3C)
bsearch:	binary search trees. tsearch, . . . . .	tsearch(3C)
tdelete, twalk: manage	bits. strip: . . . . .	strip.pdp(1)
remove symbols and relocation	bitwise boolean functions. . . . .	bool(3F)
/not, lshift, rshift: Fortran	bj: the game of black jack. . . . .	bj(6)
	black jack. . . . .	bj(6)
bj: the game of	block count of a file. . . . .	sum(1)
sum: print checksum and	block. . . . .	sync(1)
sync: update the super	boolean functions. /lshift, . . . . .	bool(3F)
rshift: Fortran bitwise	brk, sbrk: change data segment . . . . .	brk(2)
space allocation.	bs: a compiler/interpreter for . . . . .	bs(1)
modest-sized programs.	bsearch: binary search. . . . .	bsearch(3C)
	buffered input/output package. . . . .	stdio(3S)
stdio: standard	buffering to a stream. . . . .	setbuf(3S)
setbuf: assign	BX.25 link. x25alnk, . . . . .	x25alnk(3C)
x25ilnk: attach or install a	BX.25 link. . . . .	x25clnk(3C)
x25clnk: change over a	BX.25 link. x25hlnk, . . . . .	x25hlnk(3C)
x25dlnk: halt or detach a	bytes. . . . .	swab(3C)
swab: swap	C compiler. . . . .	cc(1)
cc, pcc:	C compiler for stand-alone . . . . .	scc(1)
programs. scc:	C flow graph. . . . .	cfLOW(1)
cfLOW: generate	C language preprocessor. . . . .	cpp(1)
cpp: the	C program beautifier. . . . .	cb(1)
cb:	C program checker. . . . .	lint(1)
lint: a	C program cross reference. . . . .	cxref(1)
cxref: generate	C source listing from 3B20S . . . . .	list(1)
object file. list: produce	cabs, zabs: Fortran absolute . . . . .	abs(3F)
value. abs, iabs, dabs,	cal: print calendar. . . . .	cal(1)
	calculator. . . . .	dc(1)
dc: desk	calendar. . . . .	cal(1)
cal: print	calendar: reminder service. . . . .	calendar(1)
	call another UNIX system. . . . .	cu(1C)
cu:	call. stat: . . . . .	stat(5)
data returned by stat system	calloc: main memory allocator. . . . .	malloc(3C)
malloc, free, realloc,	calls and error numbers. . . . .	intro(2)
intro: introduction to system	calls. . . . .	sys3b(2)
sys3b: 3B20S specific system	cancel: send/cancel requests . . . . .	lp(1)
to an LP line printer. lp,	card images. . . . .	pnch(4)
pnch: file format for		

XMOX

INDEX

asa: interpret ASA files.

function. cos, dcos,

commentary of an SCCS delta.

ceiling, remainder,/ floor,  
/ceil, fmod, fabs: floor,  
intrinsic/ exp, dexp,

delta: make a delta

pipe: create an interprocess

/dblc, cmplx, dcmplx, ichar,  
stream. ungetc: push  
and neqn. eqnchar: special  
user. cuserid: get  
/getchar, fgetc, getw: get  
/putchar, fputc, put  
ascii: map of ASCII

interpret ASA carriage control

\_tolower, toascii: translate

isctrl, isascii: classify

tr: translate  
directory.

constant-width text for/ cw,  
text for nroff or/ eqn, neqn,  
lint: a C program  
systems processed by fsck

formatted with the/ mm, osdd,  
file. sum: print

chess: the game of

chown,  
times: get process and  
terminate. wait: wait for

of a file.  
group.

isgraph, isctrl, isascii:  
status/ ferror, feof,

alarm: set a process's alarm

logarithm/ log, alog, dlog,  
ldclose, ldaclose:  
close:  
descriptor.  
fclose, fflush:

/real, float, singl, dblc,  
line-feeds.

comb:  
common to two sorted files.  
nice: run a  
env: set environment for  
uux: unix to unix  
system: issue a shell  
quits. nohup: run a  
net: execute a  
getopt: parse  
/shell, the standard/restricted

carriage control characters. . . . . asa(1)

cat: concatenate and print . . . . . cat(1)

cb: C program beautifier. . . . . cb(1)

cc, pcc: C compiler. . . . . cc(1)

ccos: Fortran cosine intrinsic . . . . . cos(3F)

cd: change working directory. . . . . cd(1)

cdc: change the delta . . . . . cdc(1)

ceil, fmod, fabs: floor, . . . . . floor(3M)

ceiling, remainder, absolute/ . . . . . floor(3M)

cexp: Fortran exponential . . . . . exp(3F)

cflow: generate C flow graph.  
(change) to an SCCS file. . . . . cflow(1)  
delta(1)

channel. . . . . pipe(2)

char: explicit Fortran type/  
character back into input . . . . . ftype(3F)  
ungetc(3S)

character definitions for eqn . . . . . eqnchar(5)

character login name of the . . . . . cuserid(3S)

character or word from stream. . . . . getc(3S)

character or word on a stream. . . . . putc(3S)

character set. . . . . ascii(5)

characters. asa: . . . . . asa(1)

characters. /\_toupper, . . . . . conv(3C)

characters. /isprint, isgraph, . . . . . ctype(3C)

characters. . . . . tr(1)

chdir: change working . . . . . chdir(2)

checkcw: prepare . . . . . cw(1)

checkeq: format mathematical . . . . . eqn(1)

checker. . . . . lint(1)

checklist: list of file . . . . . checklist(4)

checkmm: print/check documents . . . . . mm(1)

checksum and block count of a . . . . . sum(1)

chess. . . . . chess(6)

chess: the game of chess. . . . . chess(6)

chgrp: change owner or group. . . . . chown(1)

child process times. . . . . times(2)

child process to stop or . . . . . wait(2)

chmod: change mode. . . . . chmod(1)

chmod: change mode of file. . . . . chmod(2)

chown: change owner and group . . . . . chown(2)

chown, chgrp: change owner or . . . . . chown(1)

chroot: change root directory. . . . . chroot(2)

classify characters. /isprint, . . . . . ctype(3C)

clearerr, fileno: stream . . . . . ferror(3S)

clock. . . . . alarm(2)

clock: report CPU time used. . . . . clock(3C)

clog: Fortran natural . . . . . log(3F)

close a common object file. . . . . ldclose(3X)

close a file descriptor. . . . . close(2)

close: close a file . . . . . close(2)

close or flush a stream. . . . . fclose(3S)

cmp: compare two files. . . . . cmp(1)

cmplx, dcmplx, ichar, char:/ . . . . . ftype(3F)

col: filter reverse . . . . . col(1)

comb: combine SCCS deltas. . . . . comb(1)

combine SCCS deltas. . . . . comb(1)

comm: select or reject lines . . . . . comm(1)

command at low priority. . . . . nice(1)

command execution. . . . . env(1)

command execution. . . . . uux(1C)

command from Fortran. . . . . system(3F)

command immune to hangups and . . . . . nohup(1)

command on the PCL network. . . . . net(1C)

command options. . . . . getopt(1)

command programming language. . . . . sh(1)

and system/ timex: time a command; report process data . . . . timex(1)  
 system: issue a shell command. . . . . system(3S)  
 test: condition evaluation command. . . . . test(1)  
 time: time a command. . . . . time(1)  
 argument list(s) and execute command. xargs: construct . . . . . xargs(1)  
 getarg: return Fortran command-line argument. . . . . getarg(3F)  
 intro: introduction to commands and application/ . . . . . intro(1)  
 access graphical and numerical commands. graphics: . . . . . graphics(1G)  
 network useful with graphical commands. stat: statistical . . . . . stat(1G)  
 cdc: change the delta commentary of an SCCS delta. . . . . cdc(1)  
 ar: common archive file format. . . . . ar(4)  
 /archive files from PDP-11 to common archive format. . . . . arcv(1)  
 editor output. a.out: common assembler and link . . . . . a.out(4)  
 as: common assembler. . . . . as(1)  
 object and archive files to common formats. /convert . . . . . convert(1)  
 log10, alog10, dlog10: Fortran common logarithm intrinsic/ . . . . . log10(3F)  
 routines. ldfcn: common object file access . . . . . ldfcn(4)  
 ldopen, ldaopen: open a common object file for/ . . . . . ldopen(3X)  
 /line number entries of a common object file function. . . . . ldread(3X)  
 ldclose, ldclose: close a common object file. . . . . ldclose(3X)  
 read the file header of a common object file. ldfhread: . . . . . ldfhread(3X)  
 entries of a section of a common object file. /number . . . . . ldseek(3X)  
 the optional file header of a common object file. /seek to . . . . . ldohseek(3X)  
 /entries of a section of a common object file. . . . . ldrseek(3X)  
 /section header of a common object file. . . . . ldshread(3X)  
 an indexed/named section of a common object file. /seek to . . . . . ldsseek(3X)  
 of a symbol table entry of a common object file. /the index . . . . . ldtbindex(3X)  
 symbol table entry of a common object file. /index . . . . . ldtbread(3X)  
 seek to the symbol table of a common object file. ldtbseek: . . . . . ldtbseek(3X)  
 line number entries in a common object file. linenum: . . . . . linenum(4)  
 nm: print name list of common object file. . . . . nm(1)  
 relocation information for a common object file. reloc: . . . . . reloc(4)  
 scnhdr: section header for a common object file. . . . . scnhdr(4)  
 line number information from a common object file. /and . . . . . strip(1)  
 table format. syms: common object file symbol . . . . . syms(4)  
 filehdr: file header for common object files. . . . . filehdr(4)  
 ld: link editor for common object files. . . . . ld(1)  
 size: print section sizes of common object files. . . . . size(1)  
 comm: select or reject lines common to two sorted files. . . . . comm(1)  
 ipc: report inter-process communication facilities/ . . . . . ipc(1)  
 stdipc: standard interprocess communication package. . . . . stdipc(3C)  
 diff: differential file comparator. . . . . diff(1)  
 cmp: compare two files. . . . . cmp(1)  
 SCCS file. sccsdiff: compare two versions of an . . . . . sccsdiff(1)  
 diff3: 3-way differential file comparison. . . . . diff3(1)  
 dircmp: directory comparison. . . . . dircmp(1)  
 expression. regcmp, regex: compile and execute regular . . . . . regcmp(3X)  
 regex: regular expression compile and match routines. . . . . regex(5)  
 regcmp: regular expression compile. . . . . regcmp(1)  
 cc, pcc: C compiler. . . . . cc(1)  
 f77: Fortran 77 compiler. . . . . f77(1)  
 programs. scc: C compiler for stand-alone . . . . . scc(1)  
 yacc: yet another compiler-compiler. . . . . yacc(1)  
 modest-sized programs. bs: a compiler/interpreter for . . . . . bs(1)  
 erf, erf: error function and complementary error function. . . . . erf(3M)  
 wait: await completion of process. . . . . wait(1)  
 Fortran imaginary part of complex argument. /dimag: . . . . . aimag(3F)  
 conjg, dconjg: Fortran complex conjugate intrinsic/ . . . . . conjg(3F)  
 cprs: compress an IS25 object file. . . . . cprs(1)  
 pack, pcat, unpack: compress and expand files. . . . . pack(1)  
 table entry of a/ ldtbindex: compute the index of a symbol . . . . . ldtbindex(3X)  
 cat: concatenate and print files. . . . . cat(1)  
 synchronous printer. scat: concatenate and print files on . . . . . scat(1)  
 test: condition evaluation command. . . . . test(1)

INDEX



INDEX

conjugate intrinsic function. conjg, dconjg: Fortran complex . . . . conjg(3F)  
 an out-going terminal line connection. dial: establish . . . . dial(3C)  
 report and interactive status console. rjstat: RJE status . . . . rjstat(1C)  
     cw, checkcw: prepare constant-width text for troff. . . . cw(1)  
 execute command. xargs: construct argument list(s) and . . . . xargs(1)  
     nroff/troff, tbl, and eqn constructs. deroff: remove . . . . deroff(1)  
     ls: list contents of directories. . . . . ls(1)  
     toc: graphical table of contents routines. . . . . toc(1G)  
     csplit: context split. . . . . csplit(1)  
 asa: interpret ASA carriage control characters. . . . . asa(1)  
     ioctl: control device. . . . . ioctl(2)  
     fcntl: file control. . . . . fcntl(2)  
     msgctl: message control operations. . . . . msgctl(2)  
     semctl: semaphore control operations. . . . . semctl(2)  
     shmctl: shared memory control operations. . . . . shmctl(2)  
     fcntl: file control options. . . . . fcntl(5)  
 uucp status inquiry and job control. uustat: . . . . . uustat(1C)  
     vc: version control. . . . . vc(1)  
     terminals. term: conventional names for . . . . . term(5)  
 char: explicit Fortran type conversion. /dcmplx, ichar, . . . . ftype(3F)  
     units: conversion program. . . . . units(1)  
     dd: convert and copy a file. . . . . dd(1)  
 PDP-11 to common/ arcv: convert archive files from . . . . . arcv(1)  
 floating-point number. atof: convert ASCII string to . . . . . atof(3C)  
 integers and/ l3tol, ltol3: convert between 3-byte . . . . . l3tol(3C)  
 and base-64 ASCII/ a64l, l64a: convert between long integer . . . . a64l(3C)  
     archive files to common/ convert: convert object and . . . . . convert(1)  
     /gmtime, asctime, tzset: convert date and time to/ . . . . . ctime(3C)  
     to string. ecvt, fcvt, gcvt: convert floating-point number . . . . ecvt(3C)  
     scanf, fscanf, sscanf: convert formatted input. . . . . scanf(3S)  
     files to common/ convert: convert object and convert. . . . . convert(1)  
     strtoul, atol, atoi: convert string to integer. . . . . strtoul(3C)  
     dd: convert and copy a file. . . . . dd(1)  
     cpio: copy file archives in and out. . . . . cpio(1)  
     cp, ln, mv: copy, link or move files. . . . . cp(1)  
 uulog, uuname: unix to unix copy. uucp, . . . . . uucp(1C)  
 public UNIX-to-UNIX file copy. uuto, uupick: . . . . . uuto(1C)  
     file. core: format of core image . . . . . core(4)  
     core image file. . . . . core(4)  
     cosine intrinsic function. cos, dcosh, ccos: Fortran . . . . . cos(3F)  
     atan2: trigonometric/ sin, cos, tan, asin, acos, atan, . . . . . trig(3M)  
     hyperbolic cosine intrinsic/ cosh, dcosh: Fortran . . . . . cosh(3F)  
     functions. sinh, cosh, tanh: hyperbolic . . . . . sinh(3M)  
     cos, dcosh, ccos: Fortran cosine intrinsic function. . . . . cos(3F)  
     /dcosh: Fortran hyperbolic cosine intrinsic function. . . . . cosh(3F)  
 sum: print checksum and block count of a file. . . . . sum(1)  
     wc: word count. . . . . wc(1)  
     files. cp, ln, mv: copy, link or move . . . . . cp(1)  
     cpio: format of cpio archive. . . . . cpio(4)  
     and out. cpio: copy file archives in . . . . . cpio(1)  
     cpio: format of cpio archive. . . . . cpio(4)  
     preprocessor. cpp: the C language . . . . . cpp(1)  
     file. cprs: compress an IS25 object . . . . . cprs(1)  
     clock: report CPU time used. . . . . clock(3C)  
     craps: the game of craps. . . . . craps(6)  
     craps: the game of craps. . . . . craps(6)  
     rewrite an existing one. creat: create a new file or . . . . . creat(2)  
     file. tmpnam, tmpnam: create a name for a temporary . . . . . tmpnam(3S)  
     an existing one. creat: create a new file or rewrite . . . . . creat(2)  
     fork: create a new process. . . . . fork(2)  
     tmpfile: create a temporary file. . . . . tmpfile(3S)  
     channel. pipe: create an interprocess . . . . . pipe(2)  
     files. admin: create and administer SCCS . . . . . admin(1)

umask: set and get file creation mask. . . . . umask(2)  
 cxref: generate C program cross reference. . . . . cxref(1)  
 crypt: encode/decode. . . . . crypt(1)  
 crypt, setkey, encrypt: . . . . . crypt(3C)  
 generate DES encryption. csin: Fortran sine intrinsic . . . . . sin(3F)  
 function. sin, dsin, csplit: context split. . . . . csplit(1)  
 intrinsic/ sqrt, dsqrt, csqrt: Fortran square root . . . . . sqrt(3F)  
 terminal. ct: spawn getty to a remote . . . . . ct(1C)  
 for terminal. ctermid: generate file name . . . . . ctermid(3S)  
 asctime, tzset: convert date/ ctime, localtime, gmtime, . . . . . ctime(3C)  
 ttt, cu: call another UNIX system. . . . . cu(1C)  
 activity. sact: print cubic: tic-tac-toe. . . . . ttt(6)  
 sact(1)  
 uname: print name of current SCCS file editing . . . . . sact(1)  
 uname: get name of current UNIX system. . . . . uname(1)  
 slot in the utmp file of the current UNIX system. . . . . uname(2)  
 getcwd: get path-name of current user. /find the . . . . . tty slot(3C)  
 spline: interpolate smooth current working directory. . . . . getcwd(3C)  
 name of the user. curve. . . . . spline(1G)  
 of each line of a file. cuserid: get character login . . . . . cuserid(3S)  
 each line of a file. cut: cut out selected fields . . . . . cut(1)  
 cut: cut out selected fields . . . . . cut(1)  
 constant-width text for/ cw, checkcw: prepare . . . . . cw(1)  
 cross reference. cxref: generate C program . . . . . cxref(1)  
 absolute value. abs, iabs, dabs, cabs, zabs: Fortran . . . . . abs(3F)  
 intrinsic function. acos, dacos: Fortran arccosine . . . . . acos(3F)  
 sending daemon, line printer daemon. dpd, lpd: HONEYWELL . . . . . dpd(1C)  
 dpd, lpd: HONEYWELL sending /handle special functions of daemon, line printer daemon. . . . . dpd(1C)  
 special functions of DASI 300 and 300s terminals. . . . . 300(1)  
 intrinsic function. asin, DASI 450 terminal. /handle . . . . . 450(1)  
 /time a command; report process dasin: Fortran arcsine . . . . . asin(3F)  
 /sgetl: access long numeric data and system activity. . . . . timex(1)  
 plock: lock process, text, or data in a machine independent/ . . . . . sputl(3X)  
 prof: display profile data in memory. . . . . plock(2)  
 call. stat: data. . . . . prof(1)  
 brk, sbrk: change data returned by stat system . . . . . stat(5)  
 types: primitive system data segment space allocation. . . . . brk(2)  
 join: relational data types. . . . . types(5)  
 intrinsic function. atan, database operator. . . . . join(1)  
 intrinsic function. atan2, datan: Fortran arctangent . . . . . atan(3F)  
 /asctime, tzset: convert datan2: Fortran arctangent . . . . . atan2(3F)  
 date: print and set the date date and time to string. . . . . ctime(3C)  
 /idint, real, float, sngl, date. . . . . date(1)  
 /float, sngl, dble, cmplx, date: print and set the date. . . . . date(1)  
 conjugate intrinsic/ conjg, dble, cmplx, ichar, / ftype(3F)  
 intrinsic function. cos, dc: desk calculator. . . . . dc(1)  
 cosine intrinsic/ cosh, dcmplx, ichar, char: explicit/ . . . . . ftype(3F)  
 adb: absolute dconjg: Fortran complex . . . . . conjg(3F)  
 eqnchar: special character dcosh, ccosh: Fortran cosine . . . . . cos(3F)  
 names. basename, dirname: dcosh: Fortran hyperbolic . . . . . cosh(3F)  
 file. tail: dd: convert and copy a file. . . . . dd(1)  
 delta commentary of an SCCS debugger. . . . . adb(1)  
 file. delta: make a debugger. . . . . sdb(1)  
 delta. cdc: change the definitions for eqn and neqn. . . . . eqnchar(5)  
 delta (change) to an SCCS deliver portions of path . . . . . basename(1)  
 delta commentary of an SCCS deliver the last part of a . . . . . tail(1)  
 delta from an SCCS file. delta. cdc: change the . . . . . cdc(1)  
 delta: make a delta (change) delta (change) to an SCCS . . . . . delta(1)  
 deltas. . . . . delta(1)  
 deny messages. delta commentary of an SCCS . . . . . cdc(1)  
 deroff: remove nroff/troff, delta from an SCCS file. . . . . rmdel(1)  
 DES encryption. crypt, delta: make a delta (change) . . . . . delta(1)  
 . . . . . comb(1)  
 . . . . . mesg(1)  
 . . . . . deroff(1)  
 . . . . . crypt(3C)

-ZOMIX-

Permuted Index

system: format of 3B20S system  
     close: close a file  
 dup: duplicate an open file  
     dc:  
         x25hlnk, x25dlnk: halt or  
             file. access:  
             file:  
                 master: master  
                 master: master  
                 ioctl: control  
                 /tekset, td: graphical  
 exponential intrinsic/ exp,  
 terminal line connection.  
     ratfor: rational Fortran  
         bdiff: big  
         comparator.  
         comparison.  
         sdiff: side-by-side  
         diffmk: mark  
             diff:  
                 diff3: 3-way  
                 between files.  
 of complex argument. aimag,  
     intrinsic function. aint,  
     dir: format of  
         ls: list contents of  
 rm, rmdir: remove files or  
     cd: change working  
     chdir: change working  
     chroot: change root  
         dircmp:  
             unlink: remove  
 path-name of current working  
     mkdir: make a  
     pwd: working  
 ordinary file. mknod: make a  
     path names. basename,  
     printers. enable,  
         acct: enable or  
         dis: 3B20S  
             sadb:  
                 du: summarize  
                 prof:  
                 hypot: Euclidean  
     /lcong48: generate uniformly  
         logarithm/ log, alog,  
         logarithm/ log10, alog10,  
 max, max0, amax0, max1, amax1,  
 min, min0, amin0, min1, amin1,  
     intrinsic/ mod, amod,  
     nearest integer/ anint,  
 mm, osdd, checkmm: print/check  
 macro package for formatting  
 macro package for formatting  
     slides. mmt, mvt: typeset  
 daemon, line printer daemon.  
     reversi: a game of  
 nrand48, mrand48, jrand48,/br/>
     graph:  
     arithmetic: provide  
     description file. . . . . system(4)  
     descriptor. . . . . close(2)  
     descriptor. . . . . dup(2)  
     desk calculator. . . . . dc(1)  
     detach a BX.25 link. . . . . x25hlnk(3C)  
     determine accessibility of a . . . . . access(2)  
     determine file type. . . . . file(1)  
     device information table. . . . . master.dec(4)  
     device information table. . . . . master.u3b(4)  
     device. . . . . ioctl(2)  
     device routines and filters. . . . . gdev(1G)  
     dexp, cexp: Fortran . . . . . exp(3F)  
     dial: establish an out-going . . . . . dial(3C)  
     dialect. . . . . ratfor(1)  
     diff. . . . . bdiff(1)  
     diff: differential file . . . . . diff(1)  
     diff3: 3-way differential file . . . . . diff3(1)  
     difference program. . . . . sdiff(1)  
     differences between files. . . . . diffmk(1)  
     differential file comparator. . . . . diff(1)  
     differential file comparison. . . . . diff3(1)  
     diffmk: mark differences . . . . . diffmk(1)  
     dimag: Fortran imaginary part . . . . . aimag(3F)  
     dint: Fortran integer part . . . . . aint(3F)  
     dir: format of directories. . . . . dir(4)  
     dircmp: directory comparison. . . . . dircmp(1)  
     directories. . . . . dir(4)  
     directories. . . . . ls(1)  
     directories. . . . . rm(1)  
     directory. . . . . cd(1)  
     directory. . . . . chdir(2)  
     directory. . . . . chroot(2)  
     directory comparison. . . . . dircmp(1)  
     directory entry. . . . . unlink(2)  
     directory. getcwd: get . . . . . getcwd(3C)  
     directory. . . . . mkdir(1)  
     directory name. . . . . pwd(1)  
     directory, or a special or . . . . . mknod(2)  
     dirname: deliver portions of . . . . . basename(1)  
     dis: 3B20S disassembler. . . . . dis(1)  
     disable: enable/disable LP . . . . . enable(1)  
     disable process accounting. . . . . acct(2)  
     disassembler. . . . . dis(1)  
     disk access profiler. . . . . sadp(1)  
     disk usage. . . . . du(1)  
     display profile data. . . . . prof(1)  
     distance function. . . . . hypot(3M)  
     distributed pseudo-random/ . . . . . drand48(3C)  
     dlog, clog: Fortran natural . . . . . log(3F)  
     dlog10: Fortran common . . . . . log10(3F)  
     dmax1: Fortran maximum-value/ . . . . . max(3F)  
     dmin1: Fortran minimum-value/ . . . . . min(3F)  
     dmod: Fortran remaindering . . . . . mod(3F)  
     dnint, nint, idnint: Fortran . . . . . round(3F)  
     documents formatted with the/ . . . . . mm(1)  
     documents. mm: the MM . . . . . mm(5)  
     documents. /the OSDD adapter . . . . . mosd(5)  
     documents, view graphs, and . . . . . mmt(1)  
     dpd, lpd: HONEYWELL sending . . . . . dpd(1C)  
     dpr: off-line print. . . . . dpr(1C)  
     dramatic reversals. . . . . reversi(6)  
     drand48, erand48, lrand48, . . . . . drand48(3C)  
     draw a graph. . . . . graph(1G)  
     drill in number facts. . . . . arithmetic(6)

XIMD - 1

transfer-of-sign/ sign, isign, intrinsic function. sin, intrinsic function. sinh, root intrinsic/ sqrt, intrinsic function. tan, tangent intrinsic/ tanh,

    an object file.

        od: octal

    object file. dump:

        descriptor.

    descriptor. dup:

        duplicate an open file

        echo:

            echo arguments.

            echo: echo arguments.

floating-point number to/

    program. end, etext,

sact: print current SCCS file

    ed, red: text files. ld: link

        se: screen

    ged: graphical

        ld: link

common assembler and link

    PDP-11 assembler and link

        sed: stream

    /user, real group, and

and/ /getegid: get real user, Language.

    fsplit: split f77, ratfor, or for a pattern. grep,

enable/disable LP printers.

    accounting. acct:

        enable, disable:

            crypt:

        encryption. crypt, setkey,

    setkey, encrypt: generate DES

        makekey: generate

        locations in program.

    /getgrgid, getgrnam, setgrent,

    /getpwuid, getpwnam, setpwent,

    utmp/ /pututline, setutent,

        nlist: get

    file. linenum: line number

        man, manprog: print

    man: macros for formatting

    file/ /manipulate line number

    common/ /seek to line number

    /ldnrseek: seek to relocation

    utmp, wtmp: utmp and wtmp

        endgrent: get group file

        endpwent: get password file

        utmpname: access utmp file

    /the index of a symbol table

    /read an indexed symbol table

    putpwent: write password file

    unlink: remove directory

        command execution.

    profile: setting up an

        environ: user

        execution. env: set

    getenv: return value for

    getenv: return Fortran

    dsign: Fortran . . . . . sign(3F)

    dsin, csin: Fortran sine . . . . . sin(3F)

    dsinh: Fortran hyperbolic sine . . . . . sinh(3F)

    dsqrt, csqrt: Fortran square . . . . . sqrt(3F)

    dtan: Fortran tangent . . . . . tan(3F)

    dtanh: Fortran hyperbolic . . . . . tanh(3F)

    du: summarize disk usage. . . . . du(1)

    dump: dump selected parts of . . . . . dump(1)

    dump. . . . . od(1)

    dump selected parts of an . . . . . dump(1)

    dup: duplicate an open file . . . . . dup(2)

    duplicate an open file . . . . . dup(2)

    echo arguments. . . . . echo(1)

    echo: echo arguments. . . . . echo(1)

    ecvt, fcvt, gcvt: convert . . . . . ecvt(3C)

    ed, red: text editor. . . . . ed(1)

    edata: last locations in . . . . . end(3C)

    editing activity. . . . . sact(1)

    editor. . . . . ed(1)

    editor for common object . . . . . ld(1)

    editor for video terminals. . . . . se(1)

    editor. . . . . ged(1G)

    editor. . . . . ld.pdp(1)

    editor output. a.out: . . . . . a.out(4)

    editor output. a.out: . . . . . a.out.pdp(4)

    editor. . . . . sed(1)

    effective group IDs. . . . . getuid(2)

    effective user, real group, . . . . . getuid(2)

    efl: Extended Fortran . . . . . efl(1)

    efl files. . . . . fsplit(1)

    egrep, fgrep: search a file . . . . . grep(1)

    enable, disable: . . . . . enable(1)

    enable or disable process . . . . . acct(2)

    enable/disable LP printers. . . . . enable(1)

    encode/decode. . . . . crypt(1)

    encrypt: generate DES . . . . . crypt(3C)

    encryption. crypt, . . . . . crypt(3C)

    makekey: generate

    encryption key. . . . . makekey(1)

    end, etext, edata: last . . . . . end(3C)

    endgrent: get group file/ . . . . . getgrent(3C)

    endpwent: get password file/ . . . . . getpwent(3C)

    endutent, utmpname: access . . . . . getut(3C)

    entries from name list. . . . . nlist(3C)

    entries in a common object . . . . . linenum(4)

    entries in this manual. . . . . man(1)

    entries in this manual. . . . . man(5)

    entries of a common object . . . . . ldread(3X)

    entries of a section of a . . . . . ldseek(3X)

    entries of a section of a/ . . . . . ldseek(3X)

    entry formats. . . . . utmp(4)

    entry. /getgrnam, setgrent, . . . . . getgrent(3C)

    entry. /getpwnam, setpwent, . . . . . getpwent(3C)

    entry. /setutent, endutent, . . . . . getut(3C)

    entry of a common object file. . . . . ldtbindex(3X)

    entry of a common object file. . . . . ldtbread(3X)

    entry. . . . . putpwent(3C)

    entry. . . . . unlink(2)

    env: set environment for . . . . . env(1)

    environ: user environment. . . . . environ(5)

    environment at login time. . . . . profile(4)

    environment. . . . . environ(5)

    environment for command . . . . . env(1)

    environment name. . . . . getenv(3C)

    environment variable. . . . . getenv(3F)

INDEX

sky: obtain  
character definitions for  
remove nroff/troff, tbl, and  
mathematical text for nroff/  
definitions for eqn and neqn.  
mrand48, jrand48, / drand48,  
graphical device/ hpd,  
complementary error function.  
complementary error/ erf,  
format.  
system error/ perror,  
complementary/ erf, erfc:  
function and complementary  
sys\_errlist, sys\_nerr: system  
to system calls and  
matherr:  
errfile:  
hashcheck: find spelling  
terminal line/ dial:  
in program. end,  
hypot:  
expression. expr:  
test: condition  
execlp, execvp: execute a/  
execvp: execute/ execl, execv,  
execl, execv, execl, execve,  
network. net:  
execve, execlp, execvp:  
construct argument list(s) and  
regcmp, regex: compile and  
set environment for command  
sleep: suspend  
sleep: suspend  
monitor: prepare  
profil:  
uux: unix to unix command  
execvp: execute a/ execl,  
execute/ execl, execv, execl,  
/execv, execl, execve, execlp,  
a new file or rewrite an  
process.  
exit,  
exponential intrinsic/  
exponential, logarithm,  
pcat, unpack: compress and  
cplx, dcplx, ichar, char:  
exp, dex, cexp: Fortran  
exp, log, log10, pow, sqrt:  
expression.  
routines. regex: regular  
regcmp: regular  
expr: evaluate arguments as an  
compile and execute regular  
efl:  
greek: graphics for the  
fsplit: split  
remainder,/ floor, ceil, fmod,  
factor:  
true,  
data in a machine independent  
abort: generate an IOT  
a stream.  
ephemerides.  
eqn and neqn. /special  
eqn constructs. deroff:  
eqn, neqn, checkeq: format  
eqnchar: special character  
erand48, lrand48, nrand48,  
erase, hardcopy, tekset, td:  
erf, erfc: error function and  
erfc: error function and  
errfile: error-log file  
errno, sys\_errlist, sys\_nerr:  
error function and  
error function. /erfc:  
error messages. /errno,  
error numbers. /introduction  
error-handling function.  
error-log file format.  
errors. /hashmake, spellin,  
establish an out-going  
etext, edata: last locations  
Euclidean distance function.  
evaluate arguments as an  
evaluation command.  
execl, execv, execl, execve,  
execl, execve, execlp,  
execlp, execvp: execute a/  
execute a command on the PCL  
execute a file. /execl,  
execute command. xargs:  
execute regular expression.  
execution. env:  
execution for an interval.  
execution for interval.  
execution profile.  
execution time profile.  
execution.  
execv, execl, execve, execlp,  
execve, execlp, execvp:  
execvp: execute a file.  
existing one. creat: create  
exit, \_exit: terminate  
\_exit: terminate process.  
exp, dex, cexp: Fortran  
exp, log, log10, pow, sqrt:  
expand files. pack,  
explicit Fortran type/ /dble,  
exponential intrinsic/  
exponential, logarithm, power,  
expr: evaluate arguments as an  
expression compile and match  
expression compile.  
expression.  
expression. regcmp, regex:  
Extended Fortran Language.  
extended TTY-37 type-box.  
f77: Fortran 77 compiler.  
f77, ratfor, or efl files.  
fabs: floor, ceiling,  
factor a number.  
factor: factor a number.  
false: provide truth values.  
fashion.. /access long numeric  
fault.  
fclose, fflush: close or flush

INDEX

floating-point number/ ecvt,  
 fopen, freopen,  
 status inquiries. ferror,  
 fileno: stream status/  
 stream. fclose,  
 files from the HONEYWELL/  
 word from/ getc, getchar,  
 from the HONEYWELL/ fgetc,  
 stream. gets,  
 pattern. grep, egrep,  
 times. utime: set  
 ldfcn: common object  
 determine accessibility of a  
 hpio: HP 2645A terminal tape  
 tar: tape  
 cpio: copy  
 chmod: change mode of  
 change owner and group of a  
 diff: differential  
 diff3: 3-way differential  
 fcntl:  
 uupick: public UNIX-to-UNIX  
 core: format of core image  
 cprs: compress an IS25 object  
 umask: set and get  
 fields of each line of a  
 dd: convert and copy a  
 a delta (change) to an SCCS  
 close: close a  
 dup: duplicate an open  
 selected parts of an object  
 sact: print current SCCS  
 setgrent, endgrent: get group  
 endpwent: get password  
 utmpname: access utmp  
 putpwent: write password  
 execlp, execvp: execute a  
 grep, egrep, fgrep: search a  
 ldaopen: open a common object  
 acct: per-process accounting  
 ar: common archive  
 ar: archive  
 errfile: error-log  
 pnch:  
 intro: introduction to  
 entries of a common object  
 get: get a version of an SCCS  
 group: group  
 files. filehdr:  
 file. ldfhread: read the  
 ldohseek: seek to the optional  
 split: split a  
 issue: issue identification  
 of a member of an archive  
 close a common object  
 file header of a common object  
 a section of a common object  
 file header of a common object  
 a section of a common object  
 header of a common object  
 fcntl: file control. . . . . fcntl(2)  
 fcntl: file control options. . . . . fcntl(5)  
 fcvt, gcvt: convert . . . . . ecvt(3C)  
 fdopen: open a stream. . . . . fopen(3S)  
 feof, clearerr, fileno: stream . . . . . ferror(3S)  
 ferror, feof, clearerr, . . . . . ferror(3S)  
 fflush: close or flush a . . . . . fclose(3S)  
 fgetc, fgetc.demon: retrieve . . . . . fgetc(1C)  
 fgetc, getw: get character or . . . . . getc(3S)  
 fgetc.demon: retrieve files . . . . . fgetc(1C)  
 fgets: get a string from a . . . . . gets(3S)  
 fgrep: search a file for a . . . . . grep(1)  
 file access and modification . . . . . utime(2)  
 file access routines. . . . . ldfcn(4)  
 file. access: . . . . . access(2)  
 file archiver. . . . . hpio(1)  
 file archiver. . . . . tar(1)  
 file archives in and out. . . . . cpio(1)  
 file. . . . . chmod(2)  
 file. chown: . . . . . chown(2)  
 file comparator. . . . . diff(1)  
 file comparison. . . . . diff3(1)  
 file control. . . . . fcntl(2)  
 file control options. . . . . fcntl(5)  
 file copy. uuto, . . . . . uuto(1C)  
 file. . . . . core(4)  
 file. . . . . cprs(1)  
 file creation mask. . . . . umask(2)  
 file. cut: cut out selected . . . . . cut(1)  
 file. . . . . dd(1)  
 file. delta: make . . . . . delta(1)  
 file descriptor. . . . . close(2)  
 file descriptor. . . . . dup(2)  
 file: determine file type. . . . . file(1)  
 file. dump: dump . . . . . dump(1)  
 file editing activity. . . . . sact(1)  
 file entry. /getgrnam, . . . . . getgrent(3C)  
 file entry. /setpwent, . . . . . getpwent(3C)  
 file entry. /endutent, . . . . . getut(3C)  
 file entry. . . . . putpwent(3C)  
 file. /execv, execl, execve, . . . . . exec(2)  
 file for a pattern. . . . . grep(1)  
 file for reading. ldaopen, . . . . . ldaopen(3X)  
 file format. . . . . acct(4)  
 file format. . . . . ar(4)  
 file format. . . . . ar.pdp(4)  
 file format. . . . . errfile(4)  
 file format for card images. . . . . pnch(4)  
 file formats. . . . . intro(4)  
 file function. /line number . . . . . ldhread(3X)  
 file. . . . . get(1)  
 file. . . . . group(4)  
 file header for common object . . . . . filehdr(4)  
 file header of a common object . . . . . ldfhread(3X)  
 file header of a common object/ . . . . . ldohseek(3X)  
 file into pieces. . . . . split(1)  
 file. . . . . issue(4)  
 file. /read the archive header . . . . . ldahread(3X)  
 file. ldclose, ldoclose: . . . . . ldclose(3X)  
 file. ldfhread: read the . . . . . ldfhread(3X)  
 file. /line number entries of . . . . . ldseek(3X)  
 file. /seek to the optional . . . . . ldohseek(3X)  
 file. /relocation entries of . . . . . ldrseek(3X)  
 file. /indexed/named section . . . . . ldshread(3X)

INDEX

section of a common object	file. /to an indexed/named . . . . .	ldsseek(3X)
table entry of a common object	file. /the index of a symbol . . . . .	ldtbindex(3X)
table entry of a common object	file. /read an indexed symbol . . . . .	ldtbread(3X)
table list of a common object	file. /seek to the symbol . . . . .	ldtbseek(3X)
entries in a common object	file. linenum: line number . . . . .	linenum(4)
link: link to a	file. . . . .	link(2)
listing from 3B20S object	file. list: produce C source . . . . .	list(1)
or a special or ordinary	file. /make a directory, . . . . .	mknod(2)
ctermid: generate	file name for terminal. . . . .	ctermid(3S)
mktemp: make a unique	file name. . . . .	mktemp(3C)
change the format of a text	file. newform: . . . . .	newform(1)
name list of common object	file. nm: print . . . . .	nm(1)
/find the slot in the utmp	file of the current user. . . . .	ttyslot(3C)
one. creat: create a new	file or rewrite an existing . . . . .	creat(2)
passwd: password	file. . . . .	passwd(4)
or subsequent lines of one	file. /lines of several files . . . . .	paste(1)
/rewind, ftell: reposition a	file pointer in a stream. . . . .	fseek(3S)
lseek: move read/write	file pointer. . . . .	lseek(2)
prs: print an SCCS	file. . . . .	prs(1)
read: read from	file. . . . .	read(2)
for a common object	file. /relocation information . . . . .	reloc(4)
remove a delta from an SCCS	file. rmdel: . . . . .	rmdel(1)
bfs: big	file scanner. . . . .	bfs(1)
two versions of an SCCS	file. sccsdiff: compare . . . . .	sccsdiff(1)
sccsfile: format of SCCS	file. . . . .	sccsfile(4)
header for a common object	file. scnhdr: section . . . . .	scnhdr(4)
stat, fstat: get	file status. . . . .	stat(2)
from a common object	file. /line number information . . . . .	strip(1)
checksum and block count of a	file. sum: print . . . . .	sum(1)
syms: common object	file symbol table format. . . . .	syms(4)
volume.	file system: format of system . . . . .	fs(4)
mount: mount a	file system. . . . .	mount(2)
ustat: get	file system statistics. . . . .	ustat(2)
mnttab: mounted	file system table. . . . .	mnttab(4)
umount: unmount a	file system. . . . .	umount(2)
of 3B20S system description	file. system: format . . . . .	system(4)
fsck. checklist: list of	file systems processed by . . . . .	checklist(4)
deliver the last part of a	file. tail: . . . . .	tail(1)
tmpfile: create a temporary	file. . . . .	tmpfile(3S)
create a name for a temporary	file. tmpnam, tempnam: . . . . .	tmpnam(3S)
and modification times of a	file. touch: update access . . . . .	touch(1)
ftw: walk a	file tree. . . . .	ftw(3C)
file: determine	file type. . . . .	file(1)
undo a previous get of an SCCS	file. unget: . . . . .	unget(1)
report repeated lines in a	file. uniq: . . . . .	uniq(1)
val: validate SCCS	file. . . . .	val(1)
write: write on a	file. . . . .	write(2)
umask: set	file-creation mode mask. . . . .	umask(1)
common object files.	filehdr: file header for . . . . .	filehdr(4)
ferror, feof, clearerr,	fileno: stream status/ . . . . .	ferror(3S)
and print process accounting	file(s). acctcom: search . . . . .	acctcom(1)
create and administer SCCS	files. admin: . . . . .	admin(1)
send, gath: gather	files and/or submit RJE jobs. . . . .	send(1C)
cat: concatenate and print	files. . . . .	cat(1)
cmp: compare two	files. . . . .	cmp(1)
lines common to two sorted	files. comm: select or reject . . . . .	comm(1)
cp, ln, mv: copy, link or move	files. . . . .	cp(1)
mark differences between	files. diffmk: . . . . .	diffmk(1)
file header for common object	files. filehdr: . . . . .	filehdr(4)
find: find	files. . . . .	find(1)
archive/ arcv: convert archive	files from PDP-11 to common . . . . .	arcv(1)
fget, fget.demon: retrieve	files from the HONEYWELL 6000. . . . .	fget(1C)
format specification in text	files. fspec: . . . . .	fspec(4)
split f77, ratfor, or efl	files. fsplit: . . . . .	fsplit(1)

string, format of graphical	files. /graphical primitive . . . . .	gps(4)
link editor for common object	files. ld: . . . . .	ld(1)
scat: concatenate and print	files on synchronous printer. . . . .	scat(1)
rm, rmdir: remove	files or directories. . . . .	rm(1)
/merge same lines of several	files or subsequent lines of/ . . . . .	paste(1)
unpack: compress and expand	files. pack, pcat, . . . . .	pack(1)
pr: print	files. . . . .	pr(1)
section sizes of common object	files. size: print . . . . .	size(1)
size: print sizes of object	files. . . . .	size.pdp(1)
sort: sort and/or merge	files. . . . .	sort(1)
NSC network. nusend: send	files to another UNIX on the . . . . .	nusend(1C)
/convert object and archive	files to common formats. . . . .	convert(1)
fsend: send	files to the HONEYWELL 6000. . . . .	fsend(1C)
what: identify SCCS	files. . . . .	what(1)
greek: select terminal	filter. . . . .	greek(1)
nl: line numbering	filter. . . . .	nl(1)
col:	filter reverse line-feeds. . . . .	col(1)
graphical device routines and	filters. /tekset, td: . . . . .	gdev(1G)
tplot: graphics	filters. . . . .	tplot(1G)
find:	find files. . . . .	find(1)
hyphen:	find: find files. . . . .	find(1)
ttyname, isatty:	find hyphenated words. . . . .	hyphen(1)
object library. lorder:	find name of a terminal. . . . .	ttyname(3C)
hashmake, spellin, hashcheck:	find ordering relation for an . . . . .	lorder(1)
of the current user. ttyslot:	find spelling errors. spell, . . . . .	spell(1)
tee: pipe	find the slot in the utmp file . . . . .	ttyslot(3C)
int, ifix, idint, real,	fitting. . . . .	tee(1)
atof: convert ASCII string to	float, snpl, dble, cmplx,/ . . . . .	ftype(3F)
ecvt, fevt, gcv: convert	floating-point number. . . . .	atof(3C)
/modf: manipulate parts of	floating-point number to/ . . . . .	ecvt(3C)
floor, ceiling, remainder,/	floating-point numbers. . . . .	frexp(3C)
floor, ceil, fmod, fabs:	floor, ceil, fmod, fabs: . . . . .	floor(3M)
cflow: generate C	floor, ceiling, remainder,/ . . . . .	floor(3M)
fclose, fflush: close or	flow graph. . . . .	cflow(1)
remainder,/ floor, ceil,	flush a stream. . . . .	fclose(3S)
stream.	fmod, fabs: floor, ceiling, . . . . .	floor(3M)
	fopen, freopen, fdopen: open a . . . . .	fopen(3S)
	fork: create a new process. . . . .	fork(2)
per-process accounting file	format. acct: . . . . .	acct(4)
ar: common archive file	format. . . . .	ar(4)
from PDP-11 to common archive	format. /convert archive files . . . . .	arcv(1)
ar: archive file	format. . . . .	ar.pdp(4)
errfile: error-log file	format. . . . .	errfile(4)
pnc: file	format for card images. . . . .	pnc(4)
nroff or/ eqn, neqn, checkeq:	format mathematical text for . . . . .	eqn(1)
description file. system:	format of 3B20S system . . . . .	system(4)
newform: change the	format of a text file. . . . .	newform(1)
inode:	format of an inode. . . . .	inode(4)
core:	format of core image file. . . . .	core(4)
cpio:	format of cpio archive. . . . .	cpio(4)
dir:	format of directories. . . . .	dir(4)
/graphical primitive string,	format of graphical files. . . . .	gps(4)
scsfile:	format of SCCS file. . . . .	scsfile(4)
file system:	format of system volume. . . . .	fs(4)
files. fspec:	format specification in text . . . . .	fspec(4)
object file symbol table	format. syms: common . . . . .	syms(4)
troff. tbl:	format tables for nroff or . . . . .	tbl(1)
nroff:	format text. . . . .	nroff(1)
and archive files to common	formats. /convert object . . . . .	convert(1)
intro: introduction to file	formats. . . . .	intro(4)
wtmp: utmp and wtmp entry	formats. utmp, . . . . .	utmp(4)
scanf, fscanf, sscanf: convert	formatted input. . . . .	scanf(3S)
fprintf, sprintf: print	formatted output. printf, . . . . .	printf(3S)
/checkmm: print/check documents	formatted with the MM macros. . . . .	mm(1)



mptx: the macro package for formatting a permuted index. . . . . mptx(5)  
 mm: the MM macro package for formatting documents. . . . . mm(5)  
 OSDD adapter macro package for formatting documents. /the . . . . . mosd(5)  
 manual. man: macros for formatting entries in this . . . . . man(5)  
     f77: Fortran 77 compiler. . . . . f77(1)  
 abs, iabs, dabs, cabs, zabs: Fortran absolute value. . . . . abs(3F)  
   system/ signal: specify Fortran action on receipt of a . . . . . signal(3F)  
   function. acos, dacos: Fortran arccosine intrinsic . . . . . acos(3F)  
     function. asin, dasin: Fortran arcsine intrinsic . . . . . asin(3F)  
     function. atan2, datan2: Fortran arctangent intrinsic . . . . . atan2(3F)  
     function. atan, datan: Fortran arctangent intrinsic . . . . . atan(3F)  
 or, xor, not, lshift, rshift: Fortran bitwise boolean/ and, . . . . . bool(3F)  
   getarg: return Fortran command-line argument. . . . . getarg(3F)  
   log10, alog10, dlog10: Fortran common logarithm/ . . . . . log10(3F)  
   intrinsic/ conjg, dconjg: Fortran complex conjugate . . . . . conjg(3F)  
   function. cos, dcos, ccos: Fortran cosine intrinsic . . . . . cos(3F)  
     ratfor: rational Fortran dialect. . . . . ratfor(1)  
     getenv: return Fortran environment variable. . . . . getenv(3F)  
   function. exp, dexp, cexp: Fortran exponential intrinsic . . . . . exp(3F)  
     intrinsic/ cosh, dcosh: Fortran hyperbolic cosine . . . . . cosh(3F)  
     intrinsic/ sinh, dsinh: Fortran hyperbolic sine . . . . . sinh(3F)  
     intrinsic/ tanh, dtanh: Fortran hyperbolic tangent . . . . . tanh(3F)  
     complex/ aimag, dimag: Fortran imaginary part of . . . . . aimag(3F)  
     function. aint, dint: Fortran integer part intrinsic . . . . . aint(3F)  
     efl: Extended Fortran Language. . . . . efl(1)  
 amax0, max1, amax1, dmax1: Fortran maximum-value/ /max0, . . . . . max(3F)  
 amin0, min1, amin1, dmin1: Fortran minimum-value/ /min0, . . . . . min(3F)  
   log, alog, dlog, clog: Fortran natural logarithm/ . . . . . log(3F)  
   anint, dnint, nint, idnint: Fortran nearest integer/ . . . . . round(3F)  
   abort: terminate Fortran program. . . . . abort(3F)  
 functions. mod, amod, dmod: Fortran remaindering intrinsic . . . . . mod(3F)  
   function. sin, dsin, csin: Fortran sine intrinsic . . . . . sin(3F)  
   function. sqrt, dsqrt, csqrt: Fortran square root intrinsic . . . . . sqrt(3F)  
   len: return length of Fortran string. . . . . len(3F)  
   index: return location of Fortran substring. . . . . index(3F)  
 issue a shell command from Fortran. system: . . . . . system(3F)  
   function. tan, dtan: Fortran tangent intrinsic . . . . . tan(3F)  
   mclock: return Fortran time accounting. . . . . mclock(3F)  
   intrinsic/ sign, isign, dsign: Fortran transfer-of-sign . . . . . sign(3F)  
 /dcmplx, ichar, char: explicit Fortran type conversion. . . . . ftype(3F)  
   generator. srand, rand: Fortran uniform random-number . . . . . rand(3F)  
   formatted output. printf, fprintf, sprintf: print . . . . . printf(3S)  
   word on a/ putc, putchar, fputc, putw: put character or . . . . . putc(3S)  
   stream. puts, fputs: put a string on a . . . . . puts(3S)  
   input/output. fread, fwrite: binary . . . . . fread(3S)  
   memory allocator. malloc, free, realloc, calloc: main . . . . . malloc(3C)  
   stream. fopen, freopen, fdopen: open a . . . . . fopen(3S)  
   parts of floating-point/ frexp, ldexp, modf: manipulate . . . . . frexp(3C)  
 list: produce C source listing from 3B20S object file. . . . . list(1)  
 /and line number information from a common object file. . . . . strip(1)  
   gets, fgets: get a string from a stream. . . . . gets(3S)  
   rmdel: remove a delta from an SCCS file. . . . . rmdel(1)  
   getopt: get option letter from argument vector. . . . . getopt(3C)  
   read: read from file. . . . . read(2)  
 system: issue a shell command from Fortran. . . . . system(3F)  
   nlist: get entries from name list. . . . . nlist(3C)  
   arcv: convert archive files from PDP-11 to common archive/ . . . . . arcv(1)  
   getw: get character or word from stream. /getchar, fgets, . . . . . getc(3S)  
   /get.demon: retrieve files from the HONEYWELL 6000. . . . . fget(1C)  
   nsctorje: re-route jobs from the NSC network to RJE. . . . . nsctorje(1C)  
   getpw: get name from UID. . . . . getpw(3C)  
   formatted input. scanf, fscanf, sscanf: convert . . . . . scanf(3S)  
   of file systems processed by fsck. checklist: list . . . . . checklist(4)  
   reposition a file pointer in/ fseek, rewind, ftell: . . . . . fseek(3S)

INDEX

HONEYWELL 6000.	fsend: send files to . . . . .	fsend(1C)
text files.	fspec: format specification in . . . . .	fspec(4)
efl files.	fsplit: split f77, ratfor, or . . . . .	fsplit(1)
stat,	fstat: get file status. . . . .	stat(2)
pointer in a/ fseek, rewind,	ftell: reposition a file . . . . .	fseek(3S)
	ftw: walk a file tree. . . . .	ftw(3C)
Fortran arccosine intrinsic	function. acos, dacos: . . . . .	acos(3F)
Fortran integer part intrinsic	function. aint, dint: . . . . .	aint(3F)
error/ erf, erfc: error	function and complementary . . . . .	erf(3M)
Fortran arcsine intrinsic	function. asin, dasin: . . . . .	asin(3F)
Fortran arctangent intrinsic	function. atan2, datan2: . . . . .	atan2(3F)
Fortran arctangent intrinsic	function. atan, datan: . . . . .	atan(3F)
complex conjugate intrinsic	function. /dconjg: Fortran . . . . .	conjg(3F)
ccos: Fortran cosine intrinsic	function. cos, dcos, . . . . .	cos(3F)
hyperbolic cosine intrinsic	function. /dcosh: Fortran . . . . .	cosh(3F)
and complementary error	function. /error function . . . . .	erf(3M)
Fortran exponential intrinsic	function. exp, dexp, cexp: . . . . .	exp(3F)
gamma: log gamma	function. . . . .	gamma(3M)
hypot: Euclidean distance	function. . . . .	hypot(3M)
of a common object file	function. /line number entries . . . . .	ldlread(3X)
common logarithm intrinsic	function. /dlog10: Fortran . . . . .	log10(3F)
natural logarithm intrinsic	function. /dlog, clog: Fortran . . . . .	log(3F)
matherr: error-handling	function. . . . .	matherr(3M)
transfer-of-sign intrinsic	function. /dsign: Fortran . . . . .	sign(3F)
csin: Fortran sine intrinsic	function. sin, dsin, . . . . .	sin(3F)
hyperbolic sine intrinsic	function. /dsinh: Fortran . . . . .	sinh(3F)
Fortran square root intrinsic	function. sqrt, dsqrt, csqrt: . . . . .	sqrt(3F)
Fortran tangent intrinsic	function. tan, dtan: . . . . .	tan(3F)
hyperbolic tangent intrinsic	function. /dtanh: Fortran . . . . .	tanh(3F)
j0, j1, jn, y0, y1, yn: Bessel	functions. . . . .	bessel(3M)
Fortran bitwise boolean	functions. /lshift, rshift: . . . . .	bool(3F)
logarithm, power, square root	functions. /sqrt: exponential, . . . . .	exp(3M)
remainder, absolute value	functions. /floor, ceiling, . . . . .	floor(3M)
dmax1: Fortran maximum-value	functions. /max1, amax1, . . . . .	max(3F)
dmin1: Fortran minimum-value	functions. /min1, amin1, . . . . .	min(3F)
Fortran remaindering intrinsic	functions. mod, amod, dmod: . . . . .	mod(3F)
300, 300s: handle special	functions of DASI 300 and 300s/ . . . . .	300(1)
hp: handle special	functions of HP 2640 and/ . . . . .	hp(1)
terminal. 450: handle special	functions of the DASI 450 . . . . .	450(1)
Fortran nearest integer	functions. /nint, idnint: . . . . .	round(3F)
sinh, cosh, tanh: hyperbolic	functions. . . . .	sinh(3M)
atan, atan2: trigonometric	functions. /tan, asin, acos, . . . . .	trig(3M)
fread,	fwrite: binary input/output. . . . .	fread(3S)
jotto: secret word	game. . . . .	jotto(6)
moo: guessing	game. . . . .	moo(6)
back: the	game of backgammon. . . . .	back(6)
bj: the	game of black jack. . . . .	bj(6)
chess: the	game of chess. . . . .	chess(6)
craps: the	game of craps. . . . .	craps(6)
reversi: a	game of dramatic reversals. . . . .	reversi(6)
wump: the	game of hunt-the-wumpus. . . . .	wump(6)
intro: introduction to	games. . . . .	intro(6)
gamma: log	gamma function. . . . .	gamma(3M)
submit RJE jobs. send,	gamma: log gamma function. . . . .	gamma(3M)
jobs. send, gath:	gath: gather files and/or . . . . .	send(1C)
output to the HONEYWELL 6000.	gather files and/or submit RJE . . . . .	send(1C)
user.	gcat: send phototypesetter . . . . .	gcat(1C)
number to string. ecvt, fcvt,	gcosmail: send mail to HIS . . . . .	gcosmail(1C)
	gcvt: convert floating-point . . . . .	ecvt(3C)
	ged: graphical editor. . . . .	ged(1G)
	generate a maze. . . . .	maze(6)
abort:	generate an IOT fault. . . . .	abort(3C)
cflow:	generate C flow graph. . . . .	cflow(1)
reference. cxref:	generate C program cross . . . . .	cxref(1)

INDEX

crypt, setkey, encrypt:	generate DES encryption. . . . .	crypt(3C)
makekey:	generate encryption key. . . . .	makekey(1)
terminal. ctermid:	generate file name for . . . . .	ctermid(3S)
lexical tasks. lex:	generate programs for simple . . . . .	lex(1)
/srand48, seed48, lcong48:	generate uniformly distributed/ . . . . .	drand48(3C)
srand: simple random-number	generator. rand, . . . . .	rand(3C)
Fortran uniform random-number	generator. srand, rand: . . . . .	rand(3F)
gets, fgets:	get a string from a stream. . . . .	gets(3S)
get:	get a version of an SCCS file. . . . .	get(1)
ulimit:	get and set user limits. . . . .	ulimit(2)
the user. cuserid:	get character login name of . . . . .	cuserid(3S)
getc, getchar, fgetc, getw:	get character or word from/ . . . . .	getc(3S)
nlist:	get entries from name list. . . . .	nlist(3C)
umask: set and	get file creation mask. . . . .	umask(2)
stat, fstat:	get file status. . . . .	stat(2)
ustat:	get file system statistics. . . . .	ustat(2)
file.	get: get a version of an SCCS . . . . .	get(1)
/getgrnam, setgrent, endgrent:	get group file entry. . . . .	getgrent(3C)
getlogin:	get login name. . . . .	getlogin(3C)
logname:	get login name. . . . .	logname(1)
msgget:	get message queue. . . . .	msgget(2)
getpw:	get name from UID. . . . .	getpw(3C)
system. uname:	get name of current UNIX . . . . .	uname(2)
unset: undo a previous	get of an SCCS file. . . . .	unset(1)
argument vector. getopt:	get option letter from . . . . .	getopt(3C)
/getpwnam, setpwent, endpwent:	get password file entry. . . . .	getpwent(3C)
working directory. getcwd:	get path-name of current . . . . .	getcwd(3C)
times. times:	get process and child process . . . . .	times(2)
and/ getpid, getprp, getppid:	get process, process group, . . . . .	getpid(2)
/geteuid, getgid, getegid:	get real user, effective user,/ . . . . .	getuid(2)
semget:	get set of semaphores. . . . .	semget(2)
shmget:	get shared memory segment. . . . .	shmget(2)
tty:	get the terminal's name. . . . .	tty(1)
time:	get time. . . . .	time(2)
command-line argument.	getarg: return Fortran . . . . .	getarg(3F)
get character or word from/	getc, getchar, fgetc, getw: . . . . .	getc(3S)
character or word from/ getc,	getchar, fgetc, getw: get . . . . .	getc(3S)
current working directory.	getcwd: get path-name of . . . . .	getcwd(3C)
getuid, geteuid, getgid,	getegid: get real user,/ . . . . .	getuid(2)
environment variable.	getenv: return Fortran . . . . .	getenv(3F)
environment name.	getenv: return value for . . . . .	getenv(3C)
real user, effective/ getuid,	geteuid, getgid, getegid: get . . . . .	getuid(2)
user,/ getuid, geteuid,	getgid, getegid: get real . . . . .	getuid(2)
setgrent, endgrent: get group/	getgrent, getgrgid, getgrnam, . . . . .	getgrent(3C)
endgrent: get group/ getgrent,	getgrgid, getgrnam, setgrent, . . . . .	getgrent(3C)
get group/ getgrent, getgrgid,	getgrnam, setgrent, endgrent: . . . . .	getgrent(3C)
argument vector.	getlogin: get login name. . . . .	getlogin(3C)
getopt: get option letter from	getopt: get option letter from . . . . .	getopt(3C)
getopt: parse command options.	getopt: parse command options. . . . .	getopt(1)
getpass: read a password.	getpass: read a password. . . . .	getpass(3C)
getprp, getppid: get process,	getprp, getppid: get process, . . . . .	getpid(2)
getpid, getprp, getppid: get	getpid, getprp, getppid: get . . . . .	getpid(2)
getppid: get process, process	getppid: get process, process . . . . .	getpid(2)
getpw: get name from UID.	getpw: get name from UID. . . . .	getpw(3C)
getpwent, getpwuid, getpwnam,	getpwent, getpwuid, getpwnam, . . . . .	getpwent(3C)
getpwnam, setpwent, endpwent:	getpwnam, setpwent, endpwent: . . . . .	getpwent(3C)
getpwuid, getpwnam, setpwent,	getpwuid, getpwnam, setpwent, . . . . .	getpwent(3C)
a stream.	gets, fgets: get a string from . . . . .	gets(3S)
and terminal settings used by	getty. gettydefs: speed . . . . .	gettydefs(4)
ct: spawn	getty to a remote terminal. . . . .	ct(1C)
settings used by getty.	gettydefs: speed and terminal . . . . .	gettydefs(4)
getegid: get real user,/	getuid, geteuid, getgid, . . . . .	getuid(2)
pututline, setutent,/	getutent, getutid, getutline, . . . . .	getut(3C)
setutent, endutent,/ getutent,	getutid, getutline, pututline, . . . . .	getut(3C)

INDEX

setutent,/ getutent, getutid,  
 from/ getc, getchar, fgetc,  
 convert/ ctime, localtime,  
 setjmp, longjmp: non-local  
 string, format of graphical/  
     cflow: generate C flow  
         graph: draw a  
         sag: system activity  
 commands. graphics: access  
     /network useful with  
 /erase, hardcopy, tekset, td:  
     ged:  
         primitive string, format of  
         format of graphical/ gps:  
             routines. toc:  
             gutil:  
                 numerical commands.  
                 tplot:  
                 TTY-37 type-box. greek:  
                 plot:  
                 subroutines. plot:  
 mvt: typeset documents, view  
 package for typesetting view  
 extended TTY-37 type-box.  
     file for a pattern.  
     /user, effective user, real  
 /getppid: get process, process  
 chown, chgrp: change owner or  
     setgrent, endgrent: get  
     group:  
         setpgrp: set process  
         id: print user and  
         real group, and effective  
         setuid, setgid: set user and  
         newgrp: log in to a new  
         chown: change owner and  
         a signal to a process or a  
         update, and regenerate  
         ssignal,  
         hangman:  
         moo:  
             x25hlnk, x25dlnk:  
 DASI 300 and 300s/ 300, 300s:  
     2640 and 2621-series/ hp:  
     the DASI 450 terminal. 450:  
 nohup: run a command immune to  
 graphical device/ hpd, erase,  
 hcreate, hdestroy: manage  
 spell, hashmake, spellin,  
 find spelling errors. spell,  
 search tables. hsearch,  
 tables. hsearch, hcreate,  
 file. scnhdr: section  
     files. filehdr: file  
     file. ldhread: read the file  
     /seek to the optional file  
 /read an indexed/named section  
     ldahread: read the archive  
     getutline, pututline, . . . . . getut(3C)  
     getw: get character or word . . . . . getc(3S)  
     gmtime, asctime, tzset: . . . . . ctime(3C)  
     goto. . . . . setjmp(3C)  
     gps: graphical primitive . . . . . gps(4)  
     graph. . . . . cflow(1)  
     graph: draw a graph. . . . . graph(1G)  
     graph. . . . . graph(1G)  
     graph. . . . . sag(1G)  
     graphical and numerical . . . . . graphics(1G)  
     graphical commands. . . . . stat(1G)  
     graphical device routines and/  
     graphical editor. . . . . gdev(1G)  
     graphical files. /graphical . . . . . ged(1G)  
     graphical primitive string, . . . . . gps(4)  
     graphical table of contents . . . . . gps(4)  
     graphical utilities. . . . . toc(1G)  
     graphics: access graphical and . . . . . gutil(1G)  
     graphics filters. . . . . graphics(1G)  
     graphics for the extended . . . . . tplot(1G)  
     graphics interface. . . . . greek(5)  
     graphs, and slides. mmt, . . . . . plot(4)  
     graphs and slides. /macro . . . . . plot(3X)  
     greek: graphics for the . . . . . mmt(1)  
     greek: select terminal filter. . . . . mv(5)  
     grep, egrep, fgrep: search a . . . . . greek(5)  
     group, and effective group/ . . . . . greek(1)  
     group, and parent process IDs. . . . . grep(1)  
     group. . . . . getuid(2)  
     group file entry. /getgrnam, . . . . . getpid(2)  
     group file. . . . . group(4)  
     group: group file. . . . . group(4)  
     group ID. . . . . setpgrp(2)  
     group IDs and names. . . . . id(1)  
     group IDs. /effective user, . . . . . getuid(2)  
     group IDs. . . . . setuid(2)  
     group. . . . . newgrp(1)  
     group of a file. . . . . chown(2)  
     group of processes. /send . . . . . kill(2)  
     groups of programs. /maintain, . . . . . make(1)  
     gsignal: software signals. . . . . ssignal(3C)  
     guess the word. . . . . hangman(6)  
     guessing game. . . . . moo(6)  
     gutil: graphical utilities. . . . . gutil(1G)  
     halt or detach a BX.25 link. . . . . x25hlnk(3C)  
     handle special functions of . . . . . 300(1)  
     handle special functions of HP . . . . . hp(1)  
     handle special functions of . . . . . 450(1)  
     hangman: guess the word. . . . . hangman(6)  
     hangups and quits. . . . . nohup(1)  
     hardcopy, tekset, td: . . . . . gdev(1G)  
     hash search tables. hsearch, . . . . . hsearch(3C)  
     hashcheck: find spelling/ . . . . . spell(1)  
     hashmake, spellin, hashcheck: . . . . . spell(1)  
     hcreate, hdestroy: manage hash . . . . . hsearch(3C)  
     hdestroy: manage hash search . . . . . hsearch(3C)  
     header for a common object . . . . . scnhdr(4)  
     header for common object . . . . . filehdr(4)  
     header of a common object . . . . . ldhread(3X)  
     header of a common object/ . . . . . ldohseek(3X)  
     header of a common object/ . . . . . ldhread(3X)  
     header of a member of an/ . . . . . ldahread(3X)  
     help: ask for help. . . . . help(1)

INDEX

help: ask for  
     retrieve files from the  
     fsend: send files to the  
     phototypesetter output to the  
     printer daemon. dpd, lpd:  
     handle special functions of  
     archiver. hpio:  
     of HP 2640 and 2621-series/  
     td: graphical device routines/  
     file archiver.  
     manage hash search tables.  
     wump: the game of  
     cosh, dcosh: Fortran  
     sinh, cosh, tanh:  
     sinh, dsinh: Fortran  
     tanh, dtanh: Fortran  
  
     hyphen: find  
     function.  
     Fortran absolute value. abs,  
     /sngl, dble, cmplx, dcmplx,  
     semaphore set or shared memory  
     and names.  
     setpgrp: set process group  
     issue: issue  
     what:  
     dble, cmplx,/ int, ifix,  
     integer/ anint, dnint, nint,  
     id: print user and group  
     group, and parent process  
     group, and effective group  
     setgid: set user and group  
     sngl, dble, cmplx,/ int,  
     core: format of core  
     pnch: file format for card  
     aimag, dimag: Fortran  
     nohup: run a command  
     long numeric data in a machine  
     for formatting a permuted  
     of a/ ldtbindex: compute the  
     ptx: permuted  
     Fortran substring.  
     a common/ ldtbread: read an  
     ldshread, ldnsbread: read an  
     ldsseek, ldnsseek: seek to an  
     inittab: script for the  
     process. popen, pclose:  
     process.  
  
     inode: format of an  
     sscanf: convert formatted  
     push character back into  
     fread, fwrite: binary  
     stdio: standard buffered  
     fileno: stream status  
     uustat: uucp status  
     x25aln, x25ilnk: attach or  
     link. x25ipvc, x25rpvc:  
     sngl, dble, cmplx, dcmplx,/  
     abs: return  
     /164a: convert between long  
     nint, idnint: Fortran nearest  
     function. aint, dint: Fortran  
     atol, atoi: convert string to

help. . . . . help(1)  
 HONEYWELL 6000. /fget.demon: . . . fget(1C)  
 HONEYWELL 6000. . . . . fsend(1C)  
 HONEYWELL 6000. gcat: send . . . gcat(1C)  
 HONEYWELL sending daemon, line . . . dpd(1C)  
 HP 2640 and 2621-series/ hp: . . . hp(1)  
 HP 2645A terminal tape file . . . hpio(1)  
 hp: handle special functions . . . hp(1)  
 hpd, erase, hardcopy, tekset, . . . gdev(1G)  
 hpio: HP 2645A terminal tape . . . hpio(1)  
 hsearch, hcreate, hdestroy: . . . hsearch(3C)  
 hunt-the-wumpus. . . . . wump(6)  
 hyperbolic cosine intrinsic/ . . . cosh(3F)  
 hyperbolic functions. . . . . sinh(3M)  
 hyperbolic sine intrinsic/ . . . sinh(3F)  
 hyperbolic tangent intrinsic/ . . . tanh(3F)  
 hyphen: find hyphenated words. . . . . hyphen(1)  
 hyphenated words. . . . . hyphen(1)  
 hypot: Euclidean distance . . . . . hypot(3M)  
 iabs, dabs, cabs, zabs: . . . . . abs(3F)  
 ichar, char: explicit Fortran/ . . . . . ftype(3F)  
 id. /remove a message queue, . . . . . iperm(1)  
 id: print user and group IDs . . . . . id(1)  
 ID. . . . . setpgrp(2)  
 identification file. . . . . issue(4)  
 identify SCCS files. . . . . what(1)  
 idint, real, float, sngl, . . . . . ftype(3F)  
 idnint: Fortran nearest . . . . . round(3F)  
 IDs and names. . . . . id(1)  
 IDs. /get process, process . . . . . getpid(2)  
 IDs. /effective user, real . . . . . getuid(2)  
 IDs. setuid, . . . . . setuid(2)  
 ifix, idint, real, float, . . . . . ftype(3F)  
 image file. . . . . core(4)  
 images. . . . . pnch(4)  
 imaginary part of complex/ . . . . . aimag(3F)  
 immune to hangups and quits. . . . . nohup(1)  
 independent fashion.. /access . . . . . sputl(3X)  
 index. /the macro package . . . . . mptx(5)  
 index of a symbol table entry . . . . . ldtbindex(3X)  
 index. . . . . ptx(1)  
 index: return location of . . . . . index(3F)  
 indexed symbol table entry of . . . . . ldtbread(3X)  
 indexed/named section header/ . . . . . ldshread(3X)  
 indexed/named section of a/ . . . . . ldnsseek(3X)  
 init process. . . . . inittab(4)  
 initiate pipe to/from a . . . . . popen(3S)  
 inittab: script for the init . . . . . inittab(4)  
 inode: format of an inode. . . . . inode(4)  
 inode. . . . . inode(4)  
 input. scanf, fscanf, . . . . . scanf(3S)  
 input stream. ungetc: . . . . . ungetc(3S)  
 input/output. . . . . fread(3S)  
 input/output package. . . . . stdio(3S)  
 inquiries. /feof, clearerr, . . . . . ferror(3C)  
 inquiry and job control. . . . . uustat(1C)  
 install a BX.25 link. . . . . x25aln(3C)  
 install or remove a PVC on a . . . . . x25ipvc(3C)  
 int, ifix, idint, real, float, . . . . . ftype(3F)  
 integer absolute value. . . . . abs(3C)  
 integer and base-64 ASCII/ . . . . . a64l(3C)  
 integer functions. /dnint, . . . . . round(3F)  
 integer part intrinsic . . . . . aint(3F)  
 integer. strtol, . . . . . strtol(3C)

INDEX

/l3tol3: convert between 3-byte 3-byte integers and long rjestat: RJE status report and plot: graphics plot: graphics spline: characters. asa: sno: SNOBOL pipe: create an facilities/ ipc: report package. stdipc: standard suspend execution for an sleep: suspend execution for an acos, dacos: Fortran arccosine dint: Fortran integer part asin, dasin: Fortran arcsine datan2: Fortran arctangent datan: Fortran arctangent Fortran complex conjugate dcos, ccos: Fortran cosine Fortran hyperbolic cosine cexp: Fortran exponential Fortran common logarithm Fortran natural logarithm Fortran transfer-of-sign sin, dsin, csin: Fortran sine dsinh: Fortran hyperbolic sine csqrt: Fortran square root tan, dtan: Fortran tangent Fortran hyperbolic tangent dmod: Fortran remaindering commands and application/ formats.  miscellany. subroutines and libraries. calls and error numbers. application programs. intro: intro: intro: intro: and libraries. intro: and error numbers. intro:  abort: generate an semaphore set or shared/ communication facilities/ cprs: compress an /islower, isdigit, isxdigit, isdigit, isxdigit, isalnum,/ /isprint, isgraph, iscntrl, terminal. ttyname, /ispunct, isprint, isgraph, isalpha, isupper, islower, /isspace, ispunct, isprint, transfer-of-sign/ sign, isalnum, / isalpha, isupper, isalnum, isspace, ispunct, /isxdigit, isalnum, isspace, /isdigit, isxdigit, isalnum, Fortran. system: system: issue: file.	integers and long integers. . . . . l3tol(3C) integers. /convert between . . . . . l3tol(3C) interactive status console. . . . . rjestat(1C) interface. . . . . plot(4) interface subroutines. . . . . plot(3X) interpolate smooth curve. . . . . spline(1G) interpret ASA carriage control . . . . . asa(1) interpreter. . . . . sno(1) interprocess channel. . . . . pipe(2) inter-process communication . . . . . ipc(1) interprocess communication . . . . . stdipc(3C) interval. sleep: . . . . . sleep(1) interval. . . . . sleep(3C) intrinsic function. . . . . acos(3F) intrinsic function. aint, . . . . . aint(3F) intrinsic function. . . . . asin(3F) intrinsic function. atan2, . . . . . atan2(3F) intrinsic function. atan, . . . . . atan(3F) intrinsic function. /dconjg: . . . . . conjg(3F) intrinsic function. cos, . . . . . cos(3F) intrinsic function. /dcosh: . . . . . cosh(3F) intrinsic function. /dexp, . . . . . exp(3F) intrinsic function. /dlog10: . . . . . log10(3F) intrinsic function. /clog: . . . . . log(3F) intrinsic function. /dsign: . . . . . sign(3F) intrinsic function. . . . . sin(3F) intrinsic function. sinh, . . . . . sinh(3F) intrinsic function. /dsqrt, . . . . . sqrt(3F) intrinsic function. . . . . tan(3F) intrinsic function. /dtanh: . . . . . tanh(3F) intrinsic functions. /amod, . . . . . mod(3F) intro: introduction to . . . . . intro(1) intro: introduction to file . . . . . intro(4) intro: introduction to games. . . . . intro(6) intro: introduction to . . . . . intro(5) intro: introduction to . . . . . intro(3) intro: introduction to system . . . . . intro(2) introduction to commands and . . . . . intro(1) introduction to file formats. . . . . intro(4) intro: introduction to games. . . . . intro(6) intro: introduction to miscellany. . . . . intro(5) introduction to subroutines. . . . . intro(3) introduction to system calls . . . . . intro(2) ioctl: control device. . . . . ioctl(2) IOT fault. . . . . abort(3C) ipcrm: remove a message queue, . . . . . ipcrm(1) ipc: report inter-process . . . . . ipc(1) IS25 object file. . . . . cprs(1) isalnum, isspace, ispunct, / isalpha, isupper, islower, . . . . . ctype(3C) isascii: classify characters. . . . . ctype(3C) isatty: find name of a . . . . . ttyname(3C) iscntrl, isascii: classify / . . . . . ctype(3C) isdigit, isxdigit, isalnum, / isgraph, iscntrl, isascii: / . . . . . ctype(3C) isign, dsign: Fortran . . . . . sign(3F) islower, isdigit, isxdigit, . . . . . ctype(3C) isprint, isgraph, iscntrl, / ispunct, isprint, isgraph, / isspace, ispunct, isprint, / issue a shell command from . . . . . system(3F) issue a shell command. . . . . system(3S) issue identification file. . . . . issue(4) issue: issue identification . . . . . issue(4)
---	--

- ZDMX

isxdigit, isalnum, / isalpha, isupper, islower, isdigit, . . . . . ctype(3C)  
 /isupper, islower, isdigit, . . . . . ctype(3C)  
 news: print news items. . . . . news(1)  
     functions. j0, j1, jn, y0, y1, yn: Bessel . . . . . bessel(3M)  
     functions. j0, j1, jn, y0, y1, yn: Bessel . . . . . bessel(3M)  
 bj: the game of black jack. . . . . bj(6)  
     functions. j0, j1, jn, y0, y1, yn: Bessel . . . . . bessel(3M)  
     operator. join: relational database . . . . . join(1)  
 jotto: secret word game. . . . . jotto(6)  
 /lrand48, nrand48, mrand48, jrand48, srand48, seed48, / . . . . . drand48(3C)  
 assembler/un-assembler for/ kasb, kunb: . . . . . kasb(1)  
 makekey: generate encryption key. . . . . makekey(1)  
     process or a group of/ kill: send a signal to a . . . . . kill(2)  
     kill: terminate a process. . . . . kill(1)  
 /assembler/un-assembler for the KMC11B microprocessor. . . . . kasb(1)  
     quiz: test your knowledge. . . . . quiz(6)  
     for the KMC11B/ kasb, kunb: assembler/un-assembler . . . . . kasb(1)  
     3-byte integers and long/ l3tol, ltol3: convert between . . . . . l3tol(3C)  
     integer and base-64/ a64l, l64a: convert between long . . . . . a64l(3C)  
     scanning and processing language. awk: pattern . . . . . awk(1)  
     arbitrary-precision arithmetic language. bc: . . . . . bc(1)  
     eff: Extended Fortran Language. . . . . eff(1)  
     cpl: the C language preprocessor. . . . . cpp(1)  
     command programming language. /standard/restricted . . . . . sh(1)  
     /jrand48, srand48, seed48, lcong48: generate uniformly/ . . . . . drand48(3C)  
     object files. ld: link editor for common . . . . . ld(1)  
     ld: link editor. . . . . ld.pdp(1)  
     object file. ldclose, ldaclose: close a common . . . . . ldclose(3X)  
 header of a member of an/ ldahread: read the archive . . . . . ldahread(3X)  
 file for reading. ldopen, ldaopen: open a common object . . . . . ldopen(3X)  
     common object file. ldclose, ldaclose: close a . . . . . ldclose(3X)  
     of floating-point/ frexp, ldexp, modf: manipulate parts . . . . . frexp(3C)  
     access routines. ldfcn: common object file . . . . . ldfcn(4)  
     of a common object file. ldfhread: read the file header . . . . . ldfhread(3X)  
 line number entries/ ldread, ldinit, ldlditem: manipulate . . . . . ldread(3X)  
     number/ ldread, ldinit, ldlditem: manipulate line . . . . . ldread(3X)  
     manipulate line number/ ldread, ldinit, ldlditem: . . . . . ldread(3X)  
     number entries of a section/ ldseek, ldlnseek: seek to line . . . . . ldseek(3X)  
     entries of a section/ ldnrseek: seek to relocation . . . . . ldnrseek(3X)  
     indexed/named/ ldshread, ldnsbread: read an . . . . . ldshread(3X)  
     indexed/named/ ldsseek, ldnsseek: seek to an . . . . . ldsseek(3X)  
     file header of a common/ ldohseek: seek to the optional . . . . . ldohseek(3X)  
     object file for reading. ldopen, ldaopen: open a common . . . . . ldopen(3X)  
     relocation entries of a/ ldread, ldlnseek: seek to . . . . . ldread(3X)  
 indexed/named section header/ ldshread, ldnsbread: read an . . . . . ldshread(3X)  
     indexed/named section of a/ ldsseek, ldnsseek: seek to an . . . . . ldsseek(3X)  
     of a symbol table entry of a/ ldtbodyindex: compute the index . . . . . ldtbodyindex(3X)  
     symbol table entry of a/ ldtbodybread: read an indexed . . . . . ldtbodybread(3X)  
     table of a common object/ ldtbodyseek: seek to the symbol . . . . . ldtbodyseek(3X)  
     string. len: return length of Fortran . . . . . len(3F)  
     len: return length of Fortran string. . . . . len(3F)  
     getopt: get option letter from argument vector. . . . . getopt(3C)  
     simple lexical tasks. lex: generate programs for . . . . . lex(1)  
 generate programs for simple lexical tasks. lex: . . . . . lex(1)  
     to subroutines and libraries. /introduction . . . . . intro(3)  
     relation for an object library. /find ordering . . . . . lorder(1)  
     ar: archive and library maintainer. . . . . ar.pdp(1)  
     portable/ ar: archive and library maintainer for . . . . . ar(1)  
     ulimit: get and set user limits. . . . . ulimit(2)  
     an out-going terminal line connection. /establish . . . . . dial(3C)  
     line: read one line. . . . . line(1)  
 common object file. linenum: line number entries in a . . . . . linenum(4)  
     /ldinit, ldlditem: manipulate line number entries of a/ . . . . . ldread(3X)  
     ldseek, ldlnseek: seek to line number entries of a/ . . . . . ldseek(3X)

strip: strip symbol and  
     nl: out selected fields of each  
 lpd: HONEYWELL sending daemon,  
     send/cancel requests to an LP  
     lpr: line printer spooler. . . . . lpr(1)  
         line: read one line. . . . . line(1)  
         lsearch: linear search and update. . . . . lsearch(3C)  
         col: filter reverse  
         in a common object file.  
         files. comm: select or reject  
         uniq: report repeated  
         of several files or subsequent  
         subsequent/ paste: merge same  
         files. ld: ld(1)  
         ld: link editor. . . . . ld.pdp(1)  
         a.out: common assembler and  
         a.out: PDP-11 assembler and  
         cp, ln, mv: copy,  
         link: attach or install a BX.25  
         x25clnk: change over a BX.25  
         halt or detach a BX.25  
         install or remove a PVC on a  
         ls: nlist: get entries from name  
         nm: print name  
         nm: print name  
         by fsck. checklist:  
         from 3B20S object file.  
         file. list: produce C source  
         xargs: construct argument  
         files. cp,  
         tzset: convert date/ ctime,  
         index: return  
         end, etext, edata: last  
         memory. plock:  
         trouble:  
         natural logarithm intrinsic/  
         gamma: newgrp:  
         exponential, logarithm,/ exp,  
         common logarithm intrinsic/  
         logarithm, power,/ exp, log,  
         /alog10, dlog10: Fortran common  
         /dlog, clog: Fortran natural  
         /log10, pow, sqrt: exponential,  
         getlogin: get  
         logname: get  
         cuserid: get character  
         logname: return  
         passwd: change  
         setting up an environment at  
         user.  
         a64l, l64a: convert between  
         between 3-byte integers and  
         sputl, sgetl: access  
         setjmp,  
         for an object library.  
         nice: run a command at  
         line number information from a/  
         line numbering filter. . . . . nl(1)  
         line of a file. cut: cut . . . . . cut(1)  
         line printer daemon. dpd, . . . . . dpd(1C)  
         line printer. lp, cancel: . . . . . lp(1)  
         line printer spooler. . . . . lpr(1)  
         line: read one line. . . . . line(1)  
         lsearch: linear search and update. . . . . lsearch(3C)  
         col: filter reverse  
         in a common object file.  
         files. comm: select or reject  
         uniq: report repeated  
         of several files or subsequent  
         subsequent/ paste: merge same  
         files. ld: ld(1)  
         ld: link editor. . . . . ld.pdp(1)  
         a.out: common assembler and  
         a.out: PDP-11 assembler and  
         cp, ln, mv: copy,  
         link: attach or install a BX.25  
         x25clnk: change over a BX.25  
         halt or detach a BX.25  
         install or remove a PVC on a  
         ls: nlist: get entries from name  
         nm: print name  
         nm: print name  
         by fsck. checklist:  
         from 3B20S object file.  
         file. list: produce C source  
         xargs: construct argument  
         files. cp,  
         tzset: convert date/ ctime,  
         index: return  
         end, etext, edata: last  
         memory. plock:  
         trouble:  
         natural logarithm intrinsic/  
         gamma: newgrp:  
         exponential, logarithm,/ exp,  
         common logarithm intrinsic/  
         logarithm, power,/ exp, log,  
         /alog10, dlog10: Fortran common  
         /dlog, clog: Fortran natural  
         /log10, pow, sqrt: exponential,  
         getlogin: get  
         logname: get  
         cuserid: get character  
         logname: return  
         passwd: change  
         setting up an environment at  
         user.  
         a64l, l64a: convert between  
         between 3-byte integers and  
         sputl, sgetl: access  
         setjmp,  
         for an object library.  
         nice: run a command at  
         link editor output. . . . . a.out.pdp(4)  
         link editor output. . . . . a.out.pdp(4)  
         link: link to a file. . . . . link(2)  
         link or move files. . . . . cp(1)  
         link to a file. . . . . link(2)  
         link. x25alnk, x25ilnk: . . . . . x25alnk(3C)  
         link. . . . . x25clnk(3C)  
         link. x25hlnk, x25dlnk: . . . . . x25hlnk(3C)  
         link. x25ipvc, x25rpyc: . . . . . x25ipvc(3C)  
         lint: a C program checker. . . . . lint(1)  
         list contents of directories. . . . . ls(1)  
         list. . . . . nlist(3C)  
         list. . . . . nm.pdp(1)  
         list of common object file. . . . . nm(1)  
         list of file systems processed . . . . . checklist(4)  
         list: produce C source listing . . . . . list(1)  
         listing from 3B20S object . . . . . list(1)  
         list(s) and execute command. . . . . xargs(1)  
         ln, mv: copy, link or move . . . . . cp(1)  
         localtime, gmtime, asctime, . . . . . ctime(3C)  
         location of Fortran substring. . . . . index(3F)  
         locations in program. . . . . end(3C)  
         lock process, text, or data in . . . . . plock(2)  
         log a trouble report. . . . . trouble(1)  
         log, alog, dlog, clog: Fortran . . . . . log(3F)  
         log gamma function. . . . . gamma(3M)  
         log in to a new group. . . . . newgrp(1)  
         log, log10, pow, sqrt: . . . . . exp(3M)  
         log10, alog10, dlog10: Fortran . . . . . log10(3F)  
         log10, pow, sqrt: exponential,  
         logarithm intrinsic function. . . . . log10(3F)  
         logarithm intrinsic function. . . . . log(3F)  
         logarithm, power, square root/  
         login name. . . . . getlogin(3C)  
         login name. . . . . logname(1)  
         login name of the user. . . . . cuserid(3S)  
         login name of user. . . . . logname(3X)  
         login password. . . . . passwd(1)  
         login: sign on. . . . . login(1)  
         login time. profile: . . . . . profile(4)  
         logname: get login name. . . . . logname(1)  
         logname: return login name of . . . . . logname(3X)  
         long integer and base-64 ASCII/  
         long integers. /lto13: convert . . . . . l3tol(3C)  
         long numeric data in a machine/  
         longjmp: non-local goto. . . . . setjmp(3C)  
         lorder: find ordering relation . . . . . lorder(1)  
         low priority. . . . . nice(1)



requests to an LP line/ lp, cancel: send/cancel . . . . . lp(1)  
 send/cancel requests to an LP line printer. lp, cancel: . . . . . lp(1)  
   disable: enable/disable LP printers. enable, . . . . . enable(1)  
     lpstat: print LP status information. . . . . lpstat(1)  
 line printer daemon. dpd, lpd: HONEYWELL sending daemon, . . dpd(1C)  
   lpr: line printer spooler. . . . . lpr(1)  
   lpstat: print LP status . . . . . lpstat(1)  
 jrand48,/ drand48, erand48, lrand48, nrand48, mrand48, . . . . . drand48(3C)  
   directories. ls: list contents of . . . . . ls(1)  
   update. lsearch: linear search and . . . . . lsearch(3C)  
   pointer. lseek: move read/write file . . . . . lseek(2)  
 bitwise/ and, or, xor, not, lshift, rshift: Fortran . . . . . bool(3F)  
 integers and long/ l3tol, ltol3: convert between 3-byte . . . . . l3tol(3C)  
 /access long numeric data in a machine independent fashion.. . . . sputl(3X)  
   permuted index. mpx: the macro package for formatting a . . . . . mpx(5)  
   documents. mm: the MM macro package for formatting . . . . . mm(5)  
   mosd: the OSDD adapter macro package for formatting/ . . . . . mosd(5)  
   view graphs and/ mv: a troff macro package for typesetting . . . . . mv(5)  
     m4: macro processor. . . . . m4(1)  
   in this manual. man: macros for formatting entries . . . . . man(5)  
   formatted with the MM macros. /print/check documents . . . . . mm(1)  
   send mail to users or read mail. mail. mail, rmail: . . . . . mail(1)  
   users or read mail. mail, rmail: send mail to . . . . . mail(1)  
     gcosmail: send mail to HIS user. . . . . gcosmail(1C)  
     mail, rmail: send mail to users or read mail. . . . . mail(1)  
   malloc, free, realloc, calloc: main memory allocator. . . . . malloc(3C)  
 regenerate groups of/ make: maintain, update, and . . . . . make(1)  
   ar: archive and library maintainer. . . . . ar.pdp(1)  
   ar: archive and library maintainer for portable/ . . . . . ar(1)  
     SCCS file. delta: make a delta (change) to an . . . . . delta(1)  
     mkdir: make a directory. . . . . mkdir(1)  
   or ordinary file. mknod: make a directory, or a special . . . . . mknod(2)  
     mktemp: make a unique file name. . . . . mktemp(3C)  
   regenerate groups of/ make: maintain, update, and . . . . . make(1)  
     banner: make posters. . . . . banner(1)  
     key. makekey: generate encryption . . . . . makekey(1)  
   main memory allocator. malloc, free, realloc, calloc: . . . . . malloc(3C)  
   entries in this manual. man: macros for formatting . . . . . man(5)  
   this manual. man, manprog: print entries in . . . . . man(1)  
   tsearch, tdelete, twalk: manage binary search trees. . . . . tsearch(3C)  
   hsearch, hcreate, hdestroy: manage hash search tables. . . . . hsearch(3C)  
   of/ lldread, ldlinit, ldlitem: manipulate line number entries . . . . . lldread(3X)  
     frexp, ldexp, modf: manipulate parts of/ . . . . . frexp(3C)  
     manual. man, manprog: print entries in this . . . . . man(1)  
   manual. man, . . . . . man(1)  
   for formatting entries in this manual. man: macros . . . . . man(5)  
     ascii: map of ASCII character set. . . . . ascii(5)  
     files. diffmk: mark differences between . . . . . diffmk(1)  
   umask: set file-creation mode mask. . . . . umask(1)  
   set and get file creation mask. umask: . . . . . umask(2)  
   table. master: master device information . . . . . master.dec(4)  
   table. master: master device information . . . . . master.u3b(4)  
   information table. master: master device . . . . . master.dec(4)  
   information table. master: master device . . . . . master.u3b(4)  
 regular expression compile and match routines. regexp: . . . . . regexp(5)  
   eqn, neqn, checkeq: format mathematical text for nroff or/ . . . . . eqn(1)  
   function. matherr: error-handling . . . . . matherr(3M)  
   multiple-access-user-space/ maus: . . . . . maus(2)  
   dmax1: Fortran maximum-value/ max, max0, amax0, max1, amax1, . . . . . max(3F)  
   dmax1: Fortran/ max, max0, amax0, max1, amax1, . . . . . max(3F)  
   max, max0, amax0, max1, amax1, dmax1: Fortran/ . . . . . max(3F)  
   / max1, amax1, dmax1: Fortran maximum-value functions. . . . . max(3F)  
   maze: generate a maze. . . . . maze(6)

XMD-1

maze: generate a maze(6)  
     accounting. mlock: return Fortran time . . . . . mlock(3F)  
 memcpy, memset: memory/ memcpy, memchr, memcmp, . . . . . memory(3C)  
 memset: memory/ memcpy, memchr, memcmp, memcpy, memset: memory memory(3C)  
 operations. memcpy, memchr, memcmp, memset: memory memory(3C)  
 memcpy, memchr, memcmp, memset: memory/ . . . . . memory(3C)  
 free, realloc, calloc: main memory allocator. malloc, . . . . . malloc(3C)  
     shmctl: shared memory control operations. . . . . shmctl(2)  
 queue, semaphore set or shared memory id. /remove a message . . . . . ipcrm(1)  
     /(shared memory) operations. . . . . maus(2)  
     memcpy, memcpy, memset: memory operations. /memchr, . . . . . memory(3C)  
         shmop: shared memory operations. . . . . shmop(2)  
     lock process, text, or data in memory. plock: . . . . . plock(2)  
         shmget: get shared memory segment. . . . . shmget(2)  
     /memchr, memcmp, memcpy, memset: memory operations. . . . . memory(3C)  
         sort: sort and/or merge files. . . . . sort(1)  
     files or subsequent/ paste: merge same lines of several . . . . . paste(1)  
         msg: permit or deny messages. . . . . msg(1)  
         msgctl: message control operations. . . . . msgctl(2)  
         msgop: message operations. . . . . msgop(2)  
         msgget: get message queue. . . . . msgget(2)  
     or shared/ ipcrm: remove a message queue, semaphore set . . . . . ipcrm(1)  
         msg: permit or deny messages. . . . . msg(1)  
         sys\_nerr: system error messages. /errno, sys\_errlist, . . . . . perror(3C)  
         /for the KMC11B microprocessor. . . . . kasb(1)  
 dmin1: Fortran minimum-value/ min, min0, amin0, min1, amin1, . . . . . min(3F)  
     dmin1: Fortran/ min, min0, amin0, min1, amin1, dmin1: Fortran/ . . . . . min(3F)  
     /min1, amin1, dmin1: Fortran minimum-value functions. . . . . min(3F)  
         mkdir: make a directory. . . . . mkdir(1)  
         special or ordinary file. mknod: make a directory, or a . . . . . mknod(2)  
         name. mktemp: make a unique file . . . . . mktemp(3C)  
 formatting documents. mm: the MM macro package for . . . . . mm(5)  
     documents formatted with the MM macros. /print/check . . . . . mm(1)  
     documents formatted with the/ mm, osdd, checkmm: print/check . . . . . mm(1)  
     formatting documents. mm: the MM macro package for . . . . . mm(5)  
     view graphs, and slides. mmt, mvt: typeset documents, . . . . . mmt(1)  
         table. mnttab: mounted file system . . . . . mnttab(4)  
     remaindering intrinsic/ mod, amod, dmod: Fortran . . . . . mod(3F)  
     chmod: change mode. . . . . chmod(1)  
     umask: set file-creation mode mask. . . . . umask(1)  
     chmod: change mode of file. . . . . chmod(2)  
 bs: a compiler/interpreter for modest-sized programs. . . . . bs(1)  
     floating-point/ frexp, ldexp, modf: manipulate parts of . . . . . frexp(3C)  
     touch: update access and modification times of a file. . . . . touch(1)  
     utime: set file access and modification times. . . . . utime(2)  
     profile. monitor: prepare execution . . . . . monitor(3C)  
     package for formatting/ moo: guessing game. . . . . moo(6)  
     mount: the OSDD adapter macro . . . . . mosd(5)  
         mount: mount a file system. . . . . mount(2)  
         mount: mount a file system. . . . . mount(2)  
         mounted file system table. . . . . mnttab(4)  
         mnttab: move files. . . . . cp(1)  
     cp, ln, mv: copy, link or move read/write file pointer. . . . . lseek(2)  
         lseek: mptx: the macro package for . . . . . mptx(5)  
     formatting a permuted index. /rand48, jrand48, srand48,/ . . . . . drand48(3C)  
         operations. msgctl: message control . . . . . msgctl(2)  
         msgget: get message queue. . . . . msgget(2)  
         msgop: message operations. . . . . msgop(2)  
         multiple-access-user-space . . . . . maus(2)  
         mv: a troff macro package for . . . . . mv(5)  
         mv: copy, link or move files. . . . . cp(1)  
         mvt: typeset documents, view natural logarithm intrinsic/ . . . . . log(3F)

INDEX

/dnint, nint, idnint: Fortran nearest integer functions. . . . . round(3F)  
 mathematical text for/ eqn, neqn, checkeq: format . . . . . eqn(1)  
 definitions for eqn and neqn. /special character . . . . . eqnchar(5)  
 PCL network. net: execute a command on the . . . . . net(1C)  
 execute a command on the PCL network. net: . . . . . net(1C)  
 operation status of the NSC network. nscstat: query the . . . . . nscstat(1C)  
 to another UNIX on the NSC network. nusend: send files . . . . . nusend(1C)  
 re-route jobs from the NSC network to RJE. nscstorje: . . . . . nscstorje(1C)  
 commands. stat: statistical network useful with graphical . . . . . stat(1G)  
 a text file. newform: change the format of . . . . . newform(1)  
 news: print newgrp: log in to a new group. . . . . newgrp(1)  
 news items. . . . . news(1)  
 news: print news items. . . . . news(1)  
 nice: change priority of a . . . . . nice(2)  
 priority. nice: run a command at low . . . . . nice(1)  
 integer/ anint, dnint, nint, idnint: Fortran nearest . . . . . round(3F)  
 list. nl: line numbering filter. . . . . nl(1)  
 object file. nlist: get entries from name . . . . . nlist(3C)  
 hangups and quits. nm: print name list. . . . . nm.pdp(1)  
 setjmp, longjmp: nm: print name list of common . . . . . nm(1)  
 bitwise boolean/ and, or, xor, nohup: run a command immune to . . . . . nohup(1)  
 drand48, erand48, lrand48, non-local goto. . . . . setjmp(3C)  
 format mathematical text for not, lshift, rshift: Fortran . . . . . bool(3F)  
 tbl: format tables for nrand48, mrand48, jrand48./ drand48(3C)  
 constructs. deroff: remove nroff: format text. . . . . nroff(1)  
 the operation status of the nroff or troff. /checkeq: . . . . . eqn(1)  
 files to another UNIX on the nroff or troff. . . . . tbl(1)  
 re-route jobs from the nroff/troff, tbl, and eqn . . . . . deroff(1)  
 status of the NSC network. NSC network. nscstat: query . . . . . nscstat(1C)  
 the NSC network to RJE. NSC network. nusend: send . . . . . nusend(1C)  
 nl: line NSC network to RJE. nscstorje: . . . . . nscstorje(1C)  
 sputl, sgetl: access long nscstat: query the operation . . . . . nscstat(1C)  
 graphics: access graphical and nscstorje: re-route jobs from . . . . . nscstorje(1C)  
 UNIX on the NSC network. numbering filter. . . . . nl(1)  
 common/ convert: convert numeric data in a machine/ . . . . . sputl(3X)  
 ldfcn: common numerical commands. . . . . graphics(1G)  
 cprs: compress an IS25 nusend: send files to another . . . . . nusend(1C)  
 dump selected parts of an object and archive files to . . . . . convert(1)  
 ldopen, ldaopen: open a common object file access routines. . . . . ldfcn(4)  
 number entries of a common object file. . . . . cprs(1)  
 ldaclose: close a common object file. dump: . . . . . dump(1)  
 the file header of a common object file for reading. . . . . ldopen(3X)  
 of a section of a common object file function. /line . . . . . ldread(3X)  
 file header of a common object file. ldclose, . . . . . ldclose(3X)  
 of a section of a common object file. ldhread: read . . . . . ldhread(3X)  
 section header of a common object file. /number entries . . . . . ldseek(3X)  
 section of a common object file. /to the optional . . . . . ldohseek(3X)  
 symbol table entry of a common object file. /entries . . . . . ldrseek(3X)  
 symbol table entry of a common object file. /indexed/named . . . . . ldshread(3X)  
 the symbol table of a common object file. /indexed/named . . . . . ldsseek(3X)  
 number entries in a common object file. /the index of a . . . . . ldtbindex(3X)  
 C source listing from 3B20S object file. /read an indexed . . . . . ldtbread(3X)  
 nm: print name list of common object file. /seek to . . . . . ldtbseek(3X)  
 information for a common object file. linenum: line . . . . . linenum(4)  
 section header for a common object file. list: produce . . . . . list(1)  
 information from a common object file. . . . . nm(1)  
 format. syms: common object file. /relocation . . . . . reloc(4)  
 file header for common object file. scnhdr: . . . . . scnhdr(4)  
 ld: link editor for common object file. /and line number . . . . . strip(1)  
 print section sizes of common object file symbol table . . . . . syms(4)  
 size: print sizes of object files. filehdr: . . . . . filehdr(4)  
 object files. . . . . ld(1)  
 object files. size: . . . . . size(1)  
 object files. . . . . size.pdp(1)

find ordering relation for an	object library. lorder: . . . . .	lorder(1)
sky:	obtain ephemerides. . . . .	sky(6)
od:	octal dump. . . . .	od(1)
	od: octal dump. . . . .	od(1)
dpr:	off-line print. . . . .	dpr(1C)
reading. llopen, ldaopen:	open a common object file for . . . .	llopen(3X)
fopen, freopen, fdopen:	open a stream. . . . .	fopen(3S)
dup: duplicate an	open file descriptor. . . . .	dup(2)
open:	open for reading or writing. . . . .	open(2)
writing.	open: open for reading or . . . . .	open(2)
network. nscstat: query the	operation status of the NSC . . . . .	nscstat(1C)
/(shared memory)	operations. . . . .	maus(2)
memcmp, memcpy, memset: memory	operations. memccpy, memchr, . . . .	memory(3C)
msgctl: message control	operations. . . . .	msgctl(2)
msgop: message	operations. . . . .	msgop(2)
semctl: semaphore control	operations. . . . .	semctl(2)
semop: semaphore	operations. . . . .	semop(2)
shmctl: shared memory control	operations. . . . .	shmctl(2)
shmop: shared memory	operations. . . . .	shmop(2)
strcspn, strtok: string	operations. /strpbrk, strspn, . . . .	string(3C)
join: relational database	operator. . . . .	join(1)
vector. getopt: get	option letter from argument . . . . .	getopt(3C)
common/ lldohseek: seek to the	optional file header of a . . . . .	lldohseek(3X)
fcntl: file control	options. . . . .	fcntl(5)
stty: set the	options for a terminal. . . . .	stty(1)
getopt: parse command	options. . . . .	getopt(1)
Fortran bitwise boolean/ and,	or, xor, not, lshift, rshift: . . . . .	bool(3F)
object library. lorder: find	ordering relation for an . . . . .	lorder(1)
a directory, or a special or	ordinary file. mknod: make . . . . .	mknod(2)
formatting/ mosd: the	OSDD adapter macro package for . . . .	mosd(5)
documents formatted with/ mm,	osdd, checkmm: print/check . . . . .	mm(1)
dial: establish an	out-going terminal line/ . . . . .	dial(3C)
assembler and link editor	output. a.out: common . . . . .	a.out(4)
assembler and link editor	output. a.out: PDP-11 . . . . .	a.out.pdp(4)
sprintf: print formatted	output. printf, fprintf, . . . . .	printf(3S)
gcat: send phototypesetter	output to the HONEYWELL 6000. . . . .	gcat(1C)
chown: change	owner and group of a file. . . . .	chown(2)
chown, chgrp: change	owner or group. . . . .	chown(1)
and expand files.	pack, pcat, unpack: compress . . . . .	pack(1)
permuted/ mptx: the macro	package for formatting a . . . . .	mptx(5)
documents. mm: the MM macro	package for formatting . . . . .	mm(5)
mosd: the OSDD adapter macro	package for formatting/ . . . . .	mosd(5)
graphs and/ mv: a troff macro	package for typesetting view . . . . .	mv(5)
standard buffered input/output	package. stdio: . . . . .	stdio(3S)
interprocess communication	package. stdipc: standard . . . . .	stdipc(3C)
4014 terminal. 4014:	paginator for the Tektronix . . . . .	4014(1)
process, process group, and	parent process IDs. /get . . . . .	getpid(2)
getopt:	parse command options. . . . .	getopt(1)
	passwd: change login password. . . . .	passwd(1)
	passwd: password file. . . . .	passwd(4)
/setpwent, endpwent: get	password file entry. . . . .	getpwent(3C)
putpwent: write	password file entry. . . . .	putpwent(3C)
passwd:	password file. . . . .	passwd(4)
getpass: read a	password. . . . .	getpass(3C)
passwd: change login	password. . . . .	passwd(1)
several files or subsequent/	paste: merge same lines of . . . . .	paste(1)
dirname: deliver portions of	path names. basename, . . . . .	basename(1)
directory. getcwd: get	path-name of current working . . . . .	getcwd(3C)
fgrep: search a file for a	pattern. grep, egrep, . . . . .	grep(1)
processing language. awk:	pattern scanning and . . . . .	awk(1)
signal.	pause: suspend process until . . . . .	pause(2)
expand files. pack,	pcat, unpack: compress and . . . . .	pack(1)
cc,	pcc: C compiler. . . . .	cc(1)
net: execute a command on the	PCL network. . . . .	net(1C)

a process. popen, as: assembler for editor output. a.out: /convert archive files from truth value about your/ msg: macro package for formatting a ptx: format. acct: sys\_nerr: system error/ HONEYWELL 6000. gcat: send tc: split: split a file into channel. tee: popen, pclose: initiate data in memory. subroutines. images. ftell: reposition a file lseek: move read/write file to/from a process. and library maintainer for basename, dirname: deliver banner: make logarithm./ exp, log, log10, /sqrt: exponential, logarithm, for troff. cw, checkcw: monitor: cpp: the C language unget: undo a graphical/ gps: graphical types: prs: date: cal: of a file. sum: editing activity. sact: dpr: off-line man, manprog: cat: concatenate and scat: concatenate and pr: printf, fprintf, sprintf: lpstat: nm: object file. nm: system. uname: news: file(s). acctcom: search and object files. size: size: names. id: formatted/ mm, osdd, checkmm: HONEYWELL sending daemon, line requests to an LP line and print files on synchronous lpr: line vpr: Versatec disable: enable/disable LP print formatted output. nice: run a command at low pclose: initiate pipe to/from PDP-11. . . . . as.pdp(1) PDP-11 assembler and link . . . . . a.out.pdp(4) PDP-11 to common archive/ pdp11, u3b, u3b5, vax: provide . . . . . machid(1) permit or deny messages. . . . . msg(1) permuted index. mptx: the . . . . . mptx(5) permuted index. . . . . ptx(1) per-process accounting file . . . . . acct(4) perror, errno, sys\_errlist, . . . . . perror(3C) phototypesetter output to the . . . . . gcat(1C) phototypesetter simulator. . . . . tc(1) pieces. . . . . split(1) pipe: create an interprocess . . . . . pipe(2) pipe fitting. . . . . tee(1) pipe to/from a process. . . . . popen(3S) plock: lock process, text, or . . . . . plock(2) plot: graphics interface. . . . . plot(4) plot: graphics interface . . . . . plot(3X) pnch: file format for card . . . . . pnch(4) pointer in a stream. /rewind, . . . . . fseek(3S) pointer. . . . . lseek(2) popen, pclose: initiate pipe . . . . . popen(3S) portable archives. /archive . . . . . ar(1) portions of path names. . . . . basename(1) posters. . . . . banner(1) pow, sqrt: exponential, . . . . . exp(3M) power, square root functions. . . . . exp(3M) pr: print files. . . . . pr(1) prepare constant-width text . . . . . cw(1) prepare execution profile. . . . . monitor(3C) preprocessor. . . . . cpp(1) previous get of an SCCS file. . . . . unget(1) primitive string, format of . . . . . gps(4) primitive system data types. . . . . types(5) print an SCCS file. . . . . prs(1) print and set the date. . . . . date(1) print calendar. . . . . cal(1) print checksum and block count . . . . . sum(1) print current SCCS file . . . . . sact(1) print. . . . . dpr(1C) print entries in this manual. . . . . man(1) print files. . . . . cat(1) print files on synchronous/ . . . . . scat(1) print files. . . . . pr(1) print formatted output. . . . . printf(3S) print LP status information. . . . . lpstat(1) print name list. . . . . nm.pdp(1) print name list of common . . . . . nm(1) print name of current UNIX . . . . . uname(1) print news items. . . . . news(1) print process accounting . . . . . acctcom(1) print section sizes of common . . . . . size(1) print sizes of object files. . . . . size.pdp(1) print user and group IDs and . . . . . id(1) print/check documents . . . . . mm(1) printer daemon. dpd, lpd: . . . . . dpd(1C) printer. /cancel: send/cancel . . . . . lp(1) printer. scat: concatenate . . . . . scat(1) printer spooler. . . . . lpr(1) printer spooler. . . . . vpr(1) printers. enable, . . . . . enable(1) printf, fprintf, sprintf: . . . . . printf(3S) priority. . . . . nice(1)

nice: change  
     acct: enable or disable  
 acctcom: search and print  
     times. times: get  
 time: time a command; report  
     exit, \_exit: terminate  
     fork: create a new  
 /getpggrp, getppid: get process,  
     setpggrp: set  
     process group, and parent  
     inittab: script for the init  
     kill: terminate a  
     nice: change priority of a  
         kill: send a signal to a  
         initiate pipe to/from a  
 getpid, getpggrp, getppid: get  
     ps: report  
     memory. plock: lock  
 times: get process and child  
     wait: wait for child  
         ptrace:  
         pause: suspend  
     wait: await completion of  
         list of file systems  
     to a process or a group of  
 awk: pattern scanning and  
     m4: macro  
 provide truth value about your  
     alarm: set a  
     3B20S object file. list:  
         profile.  
         prof: display  
 monitor: prepare execution  
     profil: execution time  
     environment at login time.  
     sadb: disk access  
 standard/restricted command  
     arithmetic:  
     pdpl1, u3b, u3b5, vax:  
         true, false:  
 /generate uniformly distributed  
     stream. ungetc:  
     put character or word on a/  
     character or word on a/ putc,  
     entry.  
     stream.  
     getutent, getutid, getutline,  
     a/ putc, putchar, fputc,  
     x25rpsc: install or remove a  
     the NSC network. nscstat:  
         msgget: get message  
     ipcrm: remove a message  
     qsort:  
 command immune to hangups and  
     random-number/ srand,  
     random-number generator.

priority of a process. . . . . nice(2)  
 process accounting. . . . . acct(2)  
 process accounting file(s). . . . . acctcom(1)  
 process and child process . . . . . times(2)  
 process data and system/  
 process. . . . . exit(2)  
 process. . . . . fork(2)  
 process group, and parent/  
 process group ID. . . . . getpid(2)  
 process IDs. /get process, . . . . . setpggrp(2)  
 process. . . . . getpid(2)  
 process. . . . . inittab(4)  
 process. . . . . kill(1)  
 process. . . . . nice(2)  
 process or a group of/  
 process. popen, pclose: . . . . . kill(2)  
     popen(3S)  
 process, process group, and/  
 process status. . . . . getpid(2)  
 process, text, or data in . . . . . ps(1)  
 process times. . . . . plock(2)  
 process to stop or terminate. . . . . times(2)  
 process trace. . . . . wait(2)  
 process until signal. . . . . ptrace(2)  
 process. . . . . pause(2)  
 processed by fsck. checklist: . . . . . wait(1)  
     checklist(4)  
 processes. /send a signal . . . . . kill(2)  
 processing language. . . . . awk(1)  
 processor. . . . . m4(1)  
 processor type. /u3b5, vax: . . . . . machid(1)  
 process's alarm clock. . . . . alarm(2)  
 produce C source listing from . . . . . list(1)  
 prof: display profile data. . . . . prof(1)  
 profil: execution time . . . . . profil(2)  
 profile data. . . . . prof(1)  
 profile. . . . . monitor(3C)  
 profile. . . . . profil(2)  
 profile: setting up an . . . . . profile(4)  
 profiler. . . . . sadp(1)  
 programming language. /the . . . . . sh(1)  
 provide drill in number facts. . . . . arithmetic(6)  
 provide truth value about your/  
 provide truth values. . . . . machid(1)  
     true(1)  
 prs: print an SCCS file. . . . . prs(1)  
 ps: report process status. . . . . ps(1)  
 pseudo-random numbers. . . . . drand48(3C)  
 ptrace: process trace. . . . . ptrace(2)  
 ptx: permuted index. . . . . ptx(1)  
 push character back into input . . . . . ungetc(3S)  
 putc, putchar, fputc, putw: . . . . . putc(3S)  
 putchar, fputc, putw: put . . . . . putc(3S)  
 putpwent: write password file . . . . . putpwent(3C)  
 puts, fputs: put a string on a . . . . . puts(3S)  
 pututline, setutent, endutent,/ . . . . . getut(3C)  
 putw: put character or word on . . . . . putc(3S)  
 PVC on a link. x25ipvc, . . . . . x25ipvc(3C)  
 pwd: working directory name. . . . . pwd(1)  
 qsort: quicker sort. . . . . qsort(3C)  
 query the operation status of . . . . . nscstat(1C)  
 queue. . . . . msgget(2)  
 queue, semaphore set or shared/  
 quicker sort. . . . . ipcrm(1)  
 quits. nohup: run a . . . . . qsort(3C)  
     nohup(1)  
 quiz: test your knowledge. . . . . quiz(6)  
 rand: Fortran uniform . . . . . rand(3F)  
 rand, srand: simple . . . . . rand(3C)

INDEX

rand, srand: simple random-number generator. . . . . rand(3C)  
 srand, rand: Fortran uniform random-number generator. . . . . rand(3F)  
 fsplit: split f77, dialect. . . . . fsplit(1)  
 ratfor: ratfor: rational Fortran . . . . . ratfor(1)  
 rational Fortran dialect. . . . . ratfor(1)  
 getpass: read a password. . . . . getpass(3C)  
 entry of a common/ ldthead: read an indexed symbol table . . . . . ldthead(3X)  
 header/ ldshread, ldnshead: read an indexed/named section . . . . . ldshread(3X)  
 read: read from file. . . . . read(2)  
 rmail: send mail to users or read mail. mail, . . . . . mail(1)  
 line: read one line. . . . . line(1)  
 read: read from file. . . . . read(2)  
 member of an/ ldahread: read the archive header of a . . . . . ldahread(3X)  
 common object file. ldhread: read the file header of a . . . . . ldhread(3X)  
 open a common object file for reading. ldopen, ldaopen: . . . . . ldopen(3X)  
 open: open for reading or writing. . . . . open(2)  
 lseek: move read/write file pointer. . . . . lseek(2)  
 cmplx,/ int, ifix, idint, real, float, singl, dble, . . . . . ftype(3F)  
 allocator. malloc, free, realloc, calloc: main memory . . . . . malloc(3C)  
 specify what to do upon receipt of a signal. signal: . . . . . signal(2)  
 /specify Fortran action on receipt of a system signal. . . . . signal(3F)  
 ed, red: text editor. . . . . ed(1)  
 generate C program cross reference. cxref: . . . . . cxref(1)  
 execute regular expression. regcmp, regex: compile and . . . . . regcmp(3X)  
 compile. regcmp: regular expression . . . . . regcmp(1)  
 make: maintain, update, and regenerate groups of programs. . . . . make(1)  
 regular expression. regcmp, regex: compile and execute . . . . . regcmp(3X)  
 compile and match routines. regexp: regular expression . . . . . regexp(5)  
 match routines. regexp: regular expression compile and . . . . . regexp(5)  
 regcmp: regular expression compile. . . . . regcmp(1)  
 regex: compile and execute regular expression. regcmp, . . . . . regcmp(3X)  
 sorted files. comm: select or reject lines common to two . . . . . comm(1)  
 lorder: find ordering relation for an object/ . . . . . lorder(1)  
 join: relational database operator. . . . . join(1)  
 for a common object file. reloc: relocation information . . . . . reloc(4)  
 strip: remove symbols and relocation bits. . . . . strip.pdp(1)  
 ldrseek, ldnrseek: seek to relocation entries of a/ . . . . . ldrseek(3X)  
 common object file. reloc: relocation information for a . . . . . reloc(4)  
 /fmod, fabs: floor, ceiling, remainder, absolute value/ . . . . . floor(3M)  
 mod, amod, dmod: Fortran remaindering intrinsic/ . . . . . mod(3F)  
 calendar: reminder service. . . . . calendar(1)  
 ct: spawn getty to a remote terminal. . . . . ct(1C)  
 file. rmdel: remove a delta from an SCCS . . . . . rmdel(1)  
 semaphore set or/ ipcrm: remove a message queue, . . . . . ipcrm(1)  
 x25ipvc, x25rpsc: install or remove a PVC on a link. . . . . x25ipvc(3C)  
 unlink: remove directory entry. . . . . unlink(2)  
 rm, rmdir: remove files or directories. . . . . rm(1)  
 eqn constructs. deroff: remove nroff/troff, tbl, and . . . . . deroff(1)  
 bits. strip: remove symbols and relocation . . . . . strip.pdp(1)  
 uniq: report repeated lines in a file. . . . . uniq(1)  
 console. rjstat: RJE status report and interactive status . . . . . rjstat(1C)  
 clock: report CPU time used. . . . . clock(3C)  
 communication/ ipcsc: report inter-process . . . . . ipcsc(1)  
 timex: time a command; report process data and system/ . . . . . timex(1)  
 ps: report process status. . . . . ps(1)  
 file. uniq: report repeated lines in a . . . . . uniq(1)  
 facilities status. ststat: report synchronous terminal . . . . . ststat(1)  
 trouble: log a trouble report. . . . . trouble(1)  
 sar: system activity reporter. . . . . sar(1)  
 stream. fseek, rewind, ftell: reposition a file pointer in a . . . . . fseek(3S)  
 lp, cancel: send/cancel requests to an LP line/ . . . . . lp(1)  
 network to RJE. nsctorje: re-route jobs from the NSC . . . . . nsctorje(1C)  
 HONEYWELL/ fget, fget.demon: retrieve files from the . . . . . fget(1C)  
 argument. getarg: return Fortran command-line . . . . . getarg(3F)

variable. getenv: return Fortran environment . . . . . getenv(3F)  
 accounting. mclck: return Fortran time . . . . . mclck(3F)  
     abs: return integer absolute value. . . . . abs(3C)  
     string. len: return length of Fortran . . . . . len(3F)  
     substring. index: return location of Fortran . . . . . index(3F)  
         logname: return login name of user. . . . . logname(3X)  
     name. getenv: return value for environment . . . . . getenv(3C)  
     stat: data returned by stat system call. . . . . stat(5)  
 reversi: a game of dramatic  
     col: filter reverse line-feeds. . . . . col(1)  
     reversals. reversi: a game of dramatic . . . . . reversi(6)  
     file pointer in a/ fseek, rewind, ftell: reposition a . . . . . fseek(3S)  
     creat: create a new file or rewrite an existing one. . . . . creat(2)  
     gather files and/or submit RJE jobs. send, gath: . . . . . send(1C)  
 jobs from the NSC network RJE. nscstorje: re-route . . . . . nscstorje(1C)  
     interactive status/ rjstat: RJE status report and . . . . . rjstat(1C)  
     interactive status console. rjstat: RJE status report and . . . . . rjstat(1C)  
         directories. rm, rmdir: remove files or . . . . . rm(1)  
     read mail. mail, rmail: send mail to users or . . . . . mail(1)  
         SCCS file. rmdel: remove a delta from an . . . . . rmdel(1)  
         directories. rm, rmdir: remove files or . . . . . rm(1)  
         chroot: change root directory. . . . . chroot(2)  
     logarithm, power, square root functions. /exponential, . . . . . exp(3M)  
     /dsqrt, csqrt: Fortran square root intrinsic function. . . . . sqrt(3F)  
     /tekset, td: graphical device routines and filters. . . . . gdev(1G)  
     common object file access routines. ldfcn: . . . . . ldfcn(4)  
 expression compile and match routines. regexp: regular . . . . . regexp(5)  
     graphical table of contents routines. toc: . . . . . toc(1G)  
     standard/restricted/ sh, rsh: shell, the . . . . . sh(1)  
     and, or, xor, not, lshift, rshift: Fortran bitwise/ . . . . . bool(3F)  
         nice: run a command at low priority. . . . . nice(1)  
     hangups and quits. nohup: run a command immune to . . . . . nohup(1)  
         editing activity. sact: print current SCCS file . . . . . sact(1)  
         sadb: disk access profiler. . . . . sadp(1)  
         sag: system activity graph. . . . . sag(1G)  
         sar: system activity reporter. . . . . sar(1)  
     space allocation. brk, sbrk: change data segment . . . . . brk(2)  
         formatted input. scanf, fscanf, sscanf: convert . . . . . scanf(3S)  
         bfs: big file scanner. . . . . bfs(1)  
     language. awk: pattern scanning and processing . . . . . awk(1)  
     files on synchronous printer. scat: concatenate and print . . . . . scat(1)  
     stand-alone programs. scc: C compiler for . . . . . scc(1)  
     the delta commentary of an SCCS delta. cdc: change . . . . . cdc(1)  
         comb: combine SCCS deltas. . . . . comb(1)  
     make a delta (change) to an SCCS file. delta: . . . . . delta(1)  
         sact: print current SCCS file editing activity. . . . . sact(1)  
     get: get a version of an SCCS file. . . . . get(1)  
         prs: print an SCCS file. . . . . prs(1)  
 rmdel: remove a delta from an SCCS file. . . . . rmdel(1)  
     compare two versions of an SCCS file. scsdiff: . . . . . scsdiff(1)  
         scsfile: format of SCCS file. . . . . scsfile(4)  
     undo a previous get of an SCCS file. unget: . . . . . unget(1)  
         val: validate SCCS file. . . . . val(1)  
 admin: create and administer SCCS files. . . . . admin(1)  
     what: identify SCCS files. . . . . what(1)  
     of an SCCS file. scsdiff: compare two versions . . . . . scsdiff(1)  
         scsfile: format of SCCS file. . . . . scsfile(4)  
     common object file. scnhdr: section header for a . . . . . scnhdr(4)  
         terminals. se: screen editor for video . . . . . se(1)  
             inittab: script for the init process. . . . . inittab(4)  
             sdb: symbolic debugger. . . . . sdb(1)  
         program. sdiff: side-by-side difference . . . . . sdiff(1)  
         terminals. se: screen editor for video . . . . . se(1)  
     grep, egrep, fgrep: search a file for a pattern. . . . . grep(1)

INDEX



X11071

accounting file(s). acctcom: search and print process . . . . . acctcom(1)  
     lsearch: linear search and update. . . . . lsearch(3C)  
     bsearch: binary search. . . . . bsearch(3C)  
 hcreate, hdestroy: manage hash search tables. hsearch, . . . . . hsearch(3C)  
 tdelete, twalk: manage binary search trees. tsearch, . . . . . tsearch(3C)  
     jotto: secret word game. . . . . jotto(6)  
     object file. scnhdr: section header for a common . . . . . scnhdr(4)  
 object/ /read an indexed/named section header of a common . . . . . ldshread(3X)  
   /to line number entries of a section of a common object/ . . . . . ldlseek(3X)  
   /to relocation entries of a section of a common object/ . . . . . ldrseek(3X)  
   /seek to an indexed/named section of a common object/ . . . . . ldsseek(3X)  
     files. size: print section sizes of common object . . . . . size(1)  
     /mrand48, jrand48, srand48, sed: stream editor. . . . . sed(1)  
 section of/ ldsseek, ldnseek: seed48, lcong48: generate/ . . . . . drand48(3C)  
   a section/ ldlseek, ldnseek: seek to an indexed/named . . . . . ldsseek(3X)  
   a section/ ldrseek, ldnrseek: seek to line number entries of . . . . . ldlseek(3X)  
 header of a common/ ldohseek: seek to relocation entries of . . . . . ldrseek(3X)  
 common object file. ldtbseek: seek to the optional file . . . . . ldohseek(3X)  
 shmget: get shared memory seek to the symbol table of a . . . . . ldtbseek(3X)  
   brk, sbrk: change data segment. . . . . shmget(2)  
   to two sorted files. comm: segment space allocation. . . . . brk(2)  
     greek: select or reject lines common . . . . . comm(1)  
   of a file. cut: cut out select terminal filter. . . . . greek(1)  
   file. dump: dump selected fields of each line . . . . . cut(1)  
     semctl: semaphore control operations. . . . . semctl(2)  
     semop: semaphore operations. . . . . semop(2)  
 ipcrm: remove a message queue semaphore set or shared memory/ . . . . . ipcrm(1)  
   semget: get set of semaphores. . . . . semget(2)  
     operations. semctl: semaphore control . . . . . semctl(2)  
     semget: get set of semaphores. . . . . semget(2)  
     semop: semaphore operations. . . . . semop(2)  
     a group of processes. kill: send a signal to a process or . . . . . kill(2)  
   the NSC network. nusend: send files to another UNIX on . . . . . nusend(1C)  
     6000. fsend: send files to the HONEYWELL . . . . . fsend(1C)  
   and/or submit RJE jobs. send, gath: gather files . . . . . send(1C)  
     gcosmail: send mail to HIS user. . . . . gcosmail(1C)  
     mail. mail, rmail: send mail to users or read . . . . . mail(1)  
   the HONEYWELL 6000. gcat: send phototypesetter output to . . . . . gcat(1C)  
     line printer. lp, cancel: send/cancel requests to an LP . . . . . lp(1)  
 daemon. dpd, lpd: HONEYWELL sending daemon, line printer . . . . . dpd(1C)  
   stream. setbuf: assign buffering to a . . . . . setbuf(3S)  
     IDs. setuid, setgid: set user and group . . . . . setuid(2)  
   getgrent, getgrgid, getgrnam: setgrent, endgrent: get group/ . . . . . getgrent(3C)  
     goto. setjmp, longjmp: non-local . . . . . setjmp(3C)  
   encryption. crypt, setkey, encrypt: generate DES . . . . . crypt(3C)  
   getpwent, getpwuid, getpwnam, setpgrp: set process group ID. . . . . setpgrp(2)  
     login time. profile: setpwent, endpwent: get/ . . . . . getpwent(3C)  
   gettydefs: speed and terminal setting up an environment at . . . . . profile(4)  
     group IDs. settings used by getty. . . . . gettydefs(4)  
   /getutid, getutline, pututline, setuid, setgid: set user and . . . . . setuid(2)  
     data in a machine/ sputl, setutent, endutent, utmpname:/ . . . . . getut(3C)  
   standard/restricted command/ sgetl: access long numeric . . . . . sputl(3X)  
   operations. shmctl, sh, rsh: shell, the . . . . . sh(1)  
   queue, semaphore set or shared memory control . . . . . shmctl(2)  
   /multiple-access-user-space shared memory id. /a message . . . . . ipcrm(1)  
     shmop: shared memory operations. . . . . shmop(2)  
     shmget: get shared memory operations. . . . . shmget(2)  
   system: issue a shared memory segment. . . . . shmget(2)  
   system: issue a shell command from Fortran. . . . . system(3F)  
 command programming/ sh, rsh: shell command. . . . . system(3S)  
   operations. shell, the standard/restricted . . . . . sh(1)  
   segment. shmctl: shared memory control . . . . . shmctl(2)  
     shmget: get shared memory . . . . . shmget(2)

operations.	shmop: shared memory . . . . .	shmop(2)
program. sdiff:	side-by-side difference . . . . .	sdiff(1)
transfer-of-sign intrinsic/ login:	sign, isign, dsign: Fortran . . . . .	sign(3F)
terminal. stlogin:	sign on. . . . .	login(1)
pause: suspend process until	sign on to synchronous . . . . .	stlogin(1)
what to do upon receipt of a	signal. . . . .	pause(2)
action on receipt of a system	signal. signal: specify . . . . .	signal(2)
on receipt of a system/ upon receipt of a signal.	signal. /specify Fortran . . . . .	signal(3F)
of processes. kill: send a	signal: specify Fortran action . . . . .	signal(3F)
signal, gsignal: software	signal: specify what to do . . . . .	signal(2)
lex: generate programs for generator. rand, srand:	signal to a process or a group . . . . .	kill(2)
tc: phototypesetter	signals. . . . .	ssignal(3C)
atan, atan2: trigonometric/ intrinsic function.	simple lexical tasks. . . . .	lex(1)
sin, dsin, csin: Fortran	simple random-number . . . . .	rand(3C)
/dsinh: Fortran hyperbolic functions.	simulator. . . . .	tc(1)
hyperbolic sine intrinsic/ common object files.	sin, cos, tan, asin, acos, . . . . .	trig(3M)
files.	sin, dsin, csin: Fortran sine . . . . .	sin(3F)
size: print section size: print	sine intrinsic function. . . . .	sin(3F)
size: print	sine intrinsic function. . . . .	sinh(3F)
an interval.	sinh, cosh, tanh: hyperbolic . . . . .	sinh(3M)
interval.	sinh, dsinh: Fortran . . . . .	sinh(3F)
documents, view graphs, and typesetting view graphs and current/ ttypslot: find the spline: interpolate	size: print section sizes of . . . . .	size(1)
int, ifix, idint, real, float,	size: print sizes of object . . . . .	size.pdp(1)
sno:	sizes of common object files. . . . .	size(1)
ssignal, gsignal:	sizes of object files. . . . .	size.pdp(1)
sort:	sky: obtain ephemerides. . . . .	sky(6)
qsort: quicker	sleep: suspend execution for . . . . .	sleep(1)
tsort: topological	sleep: suspend execution for . . . . .	sleep(3C)
or reject lines common to two object file. list: produce C	slides. mmt, mvt: typeset . . . . .	mmt(1)
brk, sbrk: change data segment	slides. /macro package for . . . . .	mv(5)
terminal. ct:	slot in the utmp file of the . . . . .	ttypslot(3C)
sys3b: 3B20S	smooth curve. . . . .	spline(1G)
fspec: format	sngl, dble, cmplx, dcmplx,/ . . . . .	ftype(3F)
receipt of a system/ signal:	sno: SNOBOL interpreter. . . . .	sno(1)
receipt of a signal. signal:	SNOBOL interpreter. . . . .	sno(1)
used by getty. gettydefs:	software signals. . . . .	ssignal(3C)
hashcheck: find spelling/ spelling/ spell, hashmake,	sort and/or merge files. . . . .	sort(1)
spellin, hashcheck: find curve.	sort. . . . .	qsort(3C)
split:	sort: sort and/or merge files. . . . .	sort(1)
csplit: context	sort. . . . .	tsort(1)
files. fsplit:	sorted files. comm: select . . . . .	comm(1)
pieces.	source listing from 3B20S . . . . .	list(1)
lpr: line printer	space allocation. . . . .	brk(2)
vpr: Versatec printer	spawn getty to a remote . . . . .	ct(1C)
output. printf, fprintf,	specific system calls. . . . .	sys3b(2)
numeric data in a machine/ square root intrinsic/ power./ exp, log, log10, pow,	specification in text files. . . . .	fspec(4)
exponential, logarithm, power,	specify Fortran action on . . . . .	signal(3F)
	specify what to do upon . . . . .	signal(2)
	speed and terminal settings . . . . .	gettydefs(4)
	spell, hashmake, spellin, . . . . .	spell(1)
	spellin, hashcheck: find . . . . .	spell(1)
	spelling errors. /hashmake, . . . . .	spell(1)
	spline: interpolate smooth . . . . .	spline(1G)
	split a file into pieces. . . . .	split(1)
	split. . . . .	csplit(1)
	split f77, ratfor, or efl . . . . .	fsplit(1)
	split: split a file into . . . . .	split(1)
	spooler. . . . .	lpr(1)
	spooler. . . . .	vpr(1)
	sprintf: print formatted . . . . .	printf(3S)
	sputl, sgetl: access long . . . . .	sputl(3X)
	sqrt, dsqrt, csqrt: Fortran . . . . .	sqrt(3F)
	sqrt: exponential, logarithm, . . . . .	exp(3M)
	square root functions. /sqrt: . . . . .	exp(3M)

sqrt, dsqrt, csqrt: Fortran square root intrinsic/ . . . . . sqrt(3F)  
 random-number generator. srand, rand: Fortran uniform . . . . . rand(3F)  
     generator. rand, srand: simple random-number . . . . . rand(3C)  
 /nrand48, mrand48, jrand48, srand48, seed48, lcong48:/ . . . . . drand48(3C)  
     input. scanf, fscanf, sscanf: convert formatted . . . . . scanf(3S)  
         signals. signal, gsignal: software . . . . . signal(3C)  
     scc: C compiler for stand-alone programs. . . . . scc(1)  
         package. stdio: standard buffered input/output . . . . . stdio(3S)  
     communication/ stdipc: standard interprocess . . . . . stdipc(3C)  
         sh, rsh: shell, the standard/restricted command/ . . . . . sh(1)  
         system call. stat: data returned by stat . . . . . stat(5)  
         stat, fstat: get file status. . . . . stat(2)  
         stat: statistical network . . . . . stat(1G)  
         stat system call. . . . . stat(5)  
         statistical network useful . . . . . stat(1G)  
         statistics. . . . . ustat(2)  
     status report and interactive status console. rjstat: RJE . . . . . rjstat(1C)  
         lpstat: print LP status information. . . . . lpstat(1)  
 feof, clearerr, fileno: stream status inquiries. ferror, . . . . . ferror(3S)  
     control. ustat: uucp status inquiry and job . . . . . ustat(1C)  
     communication facilities status. /report inter-process . . . . . ipcs(1)  
     nscstat: query the operation status of the NSC network. . . . . nscstat(1C)  
         ps: report process status. . . . . ps(1)  
     status console. rjstat: RJE status report and interactive . . . . . rjstat(1C)  
         stat, fstat: get file status. . . . . stat(2)  
         terminal facilities status. /report synchronous . . . . . ststat(1)  
         input/output package. stdio: standard buffered . . . . . stdio(3S)  
         communication package. stdipc: standard interprocess . . . . . stdipc(3C)  
         sttime: set time. . . . . stime(2)  
         stlogin: sign on to . . . . . stlogin(1)  
         stop or terminate. wait: . . . . . wait(2)  
         strcat, strncat, strcmp, strcmp, strchr, strchr, strpbrk, / . . . . . string(3C)  
         strcmp, strncmp, strcmp, strcmp, strcpy, strncpy, strlen, / . . . . . string(3C)  
         strcspn, strtok: string(3C) string(3C)  
         stream editor. . . . . sed(1)  
         stream. fclose, . . . . . fclose(3S)  
         stream. . . . . fopen(3S)  
         stream. fseek, rewind, ftell: . . . . . fseek(3S)  
         stream. /getchar, fgetc, getw: . . . . . getc(3S)  
         stream. gets, . . . . . gets(3S)  
         stream. /putchar, fputc, putw: . . . . . putc(3S)  
         stream. . . . . puts(3S)  
         stream. . . . . setbuf(3S)  
         stream status inquiries. . . . . ferror(3S)  
         stream. ungetc: . . . . . ungetc(3S)  
         string. /l64a: convert between . . . . . a64l(3C)  
         string. /asctime, tzset: . . . . . ctime(3C)  
         string. /fcvt, gcvt: convert . . . . . ecvt(3C)  
         string, format of graphical/ string from a stream. . . . . gps(4)  
         string. . . . . gets(3S)  
         string. . . . . len(3F)  
         string on a stream. . . . . puts(3S)  
         string operations. /strpbrk, string to floating-point . . . . . string(3C)  
         string to integer. . . . . atof(3C)  
         strip: remove symbols and strtol(3C)  
         strip: strip symbol and line strip(1)  
         strip symbol and line number strip(1)  
         strlen, strchr, strchr, / . . . . . string(3C)  
         strncat, strcmp, strncmp, strcmp, strcmp, strncpy, / . . . . . string(3C)  
         strncpy, strlen, strchr, / . . . . . string(3C)

INDEX

/strlen, strchr, strrchr, strpbrk, strspn, strcspn,/  
 /strncpy, strlen, strchr, strrrchr, strpbrk, strspn,/  
 /strchr, strrchr, strpbrk, strspn, strcspn, strtok:/ . . . . . string(3C)  
 /strpbrk, strspn, strcspn, strtok: string operations. . . . . string(3C)  
 string to integer. . . . . strtol(3C)  
 terminal facilities status. . . . . ststat(1)  
 terminal. . . . . stty(1)  
 another user. . . . . su(1)  
 gath: gather files and/or submit RJE jobs. send, . . . . . send(1C)  
 intro: introduction to subroutines and libraries. . . . . intro(3)  
 plot: graphics interface subroutines. . . . . plot(3X)  
 /same lines of several files or subsequent lines of one file. . . . . paste(1)  
 return location of Fortran substring. index: . . . . . index(3F)  
 count of a file. . . . . sum(1)  
 du: summarize disk usage. . . . . du(1)  
 sync: update the super-block. . . . . sync(1)  
 sync: update super-block. . . . . sync(2)  
 su: become super-user or super-user or another user. . . . . su(1)  
 interval. sleep: suspend execution for an . . . . . sleep(1)  
 interval. sleep: suspend execution for . . . . . sleep(3C)  
 pause: suspend process until signal. . . . . pause(2)  
 swab: swap bytes. . . . . swab(3C)  
 swab: swap bytes. . . . . swab(3C)  
 information from/ strip: strip symbol and line number . . . . . strip(1)  
 object/ /compute the index of a symbol table entry of a common . . . . . ldtbindex(3X)  
 ldtbread: read an indexed symbol table entry of a common/ . . . . . ldtbread(3X)  
 syms: common object file symbol table format. . . . . syms(4)  
 object/ ldtbseek: seek to the symbol table of a common . . . . . ldtbseek(3X)  
 sdb: symbolic debugger. . . . . sdb(1)  
 strip: remove symbols and relocation bits. . . . . strip.pdp(1)  
 symbol table format. syms: common object file . . . . . syms(4)  
 sync: update super-block. . . . . sync(2)  
 sync: update the super block. . . . . sync(1)  
 concatenate and print files on synchronous printer. scat: . . . . . scat(1)  
 facilities/ ststat: report synchronous terminal . . . . . ststat(1)  
 stlogin: sign on to synchronous terminal. . . . . stlogin(1)  
 calls. . . . . sys3b(2)  
 error/ perror, errno, sys\_errlist, sys3b: 3B20S specific system . . . . . sys\_errlist, sys\_nerr: system . . . . . perror(3C)  
 perror, errno, sys\_errlist, sys\_nerr: system error/ . . . . . perror(3C)  
 /compute the index of a symbol table entry of a common object/ . . . . . ldtbindex(3X)  
 file. /read an indexed symbol table entry of a common object . . . . . ldtbread(3X)  
 common object file symbol table format. syms: . . . . . syms(4)  
 master device information table. master: . . . . . master.dec(4)  
 master device information table. master: . . . . . master.u3b(4)  
 mnttab: mounted file system table. . . . . mnttab(4)  
 ldtbseek: seek to the symbol table of a common object file. . . . . ldtbseek(3X)  
 toc: graphical table of contents routines. . . . . toc(1G)  
 tbl: format tables for nroff or troff. . . . . tbl(1)  
 hdestroy: manage hash search tables. hsearch, hcreate, . . . . . hsearch(3C)  
 tabs: set tabs on a terminal. . . . . tabs(1)  
 a file. tabs: set tabs on a terminal. . . . . tabs(1)  
 trigonometric/ sin, cos, tail: deliver the last part of . . . . . tail(1)  
 intrinsic function. tan, asin, acos, atan, atan2: . . . . . trig(3M)  
 tan, dtan: Fortran tangent tan(3F)  
 /dtanh: Fortran hyperbolic tangent intrinsic function. . . . . tan(3F)  
 hyperbolic tangent intrinsic/ tanh, dtanh: Fortran . . . . . tanh(3F)  
 sinh, cosh, tanh: hyperbolic functions. . . . . sinh(3M)  
 hpio: HP 2645A terminal tape file archiver. . . . . hpio(1)  
 tar: tape file archiver. . . . . tar(1)  
 programs for simple lexical tar: tape file archiver. . . . . tar(1)  
 deroff: remove nroff/troff, tasks. lex: generate . . . . . lex(1)  
 or troff. tbl, and eqn constructs. . . . . deroff(1)  
 tbl: format tables for nroff . . . . . tbl(1)

INDEX

INDEX

hpd, erase, hardcopy, tekset, search trees. tsearch,	tc: phototypesetter simulator. . . . .	tc(1)
hpd, erase, hardcopy, 4014: paginator for the temporary file. tmpnam, tmpfile: create a tmpnam: create a name for a terminals. for the Tektronix 4014 functions of the DASI 450 ct: spawn getty to a remote generate file name for ststat: report synchronous greek: select dial: establish an out-going getty. gettydefs: speed and sign on to synchronous stty: set the options for a tabs: set tabs on a hpio: HP 2645A isatty: find name of a functions of DASI 300 and 300s of HP 2640 and 2621-series tty: get the se: screen editor for video term: conventional names for kill: abort: exit, _exit: for child process to stop or command. quiz: ed, red: change the format of a fspec: format specification in /checkeq: format mathematical prepare constant-width nroff: format plock: lock process, troff: typeset ttt, cubic: data and system/ timex: time: mclock: return Fortran profil: execution up an environment at login stime: set time: get tzset: convert date and clock: report CPU process times. update access and modification get process and child process file access and modification process data and system/ file. for a temporary file. /tolower, _toupper, _tolower, contents routines. popen, pclose: initiate pipe	td: graphical device routines/ tdelete, twalk: manage binary tee: pipe fitting. . . . . teksset, td: graphical device/ Tektronix 4014 terminal. . . . . tempnam: create a name for a temporary file. . . . . temporary file. tmpnam, . . . . . term: conventional names for terminal. 4014: paginator . . . . . terminal. 450: handle special terminal. . . . . terminal. ctermid: . . . . . terminal facilities status. . . . . terminal filter. . . . . terminal line connection. . . . . terminal settings used by . . . . . terminal. stlogin: . . . . . terminal. . . . . terminal. . . . . terminal tape file archiver. terminal. ttyname, . . . . . terminals. /handle special . . . . . terminals. /special functions . . . . . terminal's name. . . . . terminals. . . . . terminals. . . . . terminate a process. . . . . terminate Fortran program. . . . . terminate process. . . . . terminate. wait: wait . . . . . test: condition evaluation . . . . . test your knowledge. . . . . text editor. . . . . text file. newform: . . . . . text files. . . . . text for nroff or troff. . . . . text for troff. cw, checkcw: . . . . . text. . . . . text, or data in memory. . . . . text. . . . . tic-tac-toe. . . . . time a command; report process time a command. . . . . time accounting. . . . . time: get time. . . . . time profile. . . . . time. profil: setting . . . . . time. . . . . time: time a command. . . . . time. . . . . time to string. /asctime, . . . . . time used. . . . . times: get process and child times of a file. touch: . . . . . times. times: . . . . . times. utime: set . . . . . timex: time a command; report tmpfile: create a temporary tmpnam, tempnam: create a name toascii: translate characters. . . . . toc: graphical table of . . . . . to/from a process. . . . .	gdev(1G) tsearch(3C) tee(1) gdev(1G) 4014(1) tmpnam(3S) tmpfile(3S) tmpnam(3S) term(5) 4014(1) 450(1) ct(1C) ctermid(3S) ststat(1) greek(1) dial(3C) gettydefs(4) stlogin(1) stty(1) tabs(1) hpio(1) ttyname(3C) 300(1) hp(1) tty(1) se(1) term(5) kill(1) abort(3F) exit(2) wait(2) test(1) quiz(6) ed(1) newform(1) fspec(4) eqn(1) cw(1) nroff(1) plock(2) troff(1) ttt(6) timex(1) time(1) mclock(3F) time(2) profil(2) profile(4) stime(2) time(1) time(2) ctime(3C) clock(3C) times(2) touch(1) times(2) utime(2) timex(1) tmpfile(3S) tmpnam(3S) conv(3C) toc(1G) popen(3S)

- topupper, tolower, \_toupper, toascii: translate/ toupper, toascii: translate/ toupper, tsort: topological sort. . . . . tsort(1)
  - modification times of a file. touch: update access and \_toupper, \_tolower, toascii: toupper, tolower, \_toupper, \_tolower, toascii: translate/ translate characters. . . . . tr(1)
  - ptrace: process sign, isign, dsign: Fortran /\_toupper, \_tolower, toascii: tr: ftw: walk a file
  - twalk: manage binary search tan, asin, acos, atan, atan2: constant-width text for mathematical text for nroff or typesetting view graphs/ mv: a format tables for nroff or
  - trouble: log a values.
  - pdp11, u3b, u3b5, vax: provide true, false: provide manage binary search trees.
  - graphics for the extended a terminal.
  - utmp file of the current/ trees. tsearch, tdelete, ichar, char: explicit Fortran file: determine file value about your processor for the extended TTY-37 types.
  - types: primitive system data graphs, and slides. mmt, mvt: troff: mv: a troff macro package for /localtime, gmtime, asctime, value about your/ pdp11, about your/ pdp11, u3b, getpw: get name from limits.
  - creation mask.
  - mask.
  - UNIX system.
  - UNIX system.
  - file. unget: an SCCS file.
  - into input stream.
  - srand, rand: Fortran /seed48, lcong48: generate a file.
  - mktemp: make a
  - uuto, uupick: public entry.
  - umount: files. pack, pcat,
  - \_tolower, toascii: translate/ . . . . . conv(3C)
  - tolower, \_toupper, \_tolower, . . . . . conv(3C)
  - topological sort. . . . . tsort(1)
  - touch: update access and . . . . . touch(1)
  - \_toupper, \_tolower, toascii: . . . . . conv(3C)
  - toupper, tolower, \_toupper, . . . . . conv(3C)
  - tplot: graphics filters. . . . . tplot(1G)
  - tr: translate characters. . . . . tr(1)
  - trace. . . . . ptrace(2)
  - transfer-of-sign intrinsic/ . . . . . sign(3F)
  - translate characters. . . . . conv(3C)
  - translate characters. . . . . tr(1)
  - tree. . . . . ftw(3C)
  - trees. tsearch, tdelete, . . . . . tsearch(3C)
  - trigonometric functions. /cos, . . . . . trig(3M)
  - troff. cw, checkcw: prepare . . . . . cw(1)
  - troff. /neqn, checkeq: format . . . . . eqn(1)
  - troff macro package for . . . . . mv(5)
  - troff. tbl: . . . . . tbl(1)
  - troff: typeset text. . . . . troff(1)
  - trouble: log a trouble report. . . . . trouble(1)
  - true, false: provide truth . . . . . true(1)
  - truth value about your/ . . . . . machid(1)
  - truth values. . . . . true(1)
  - tsearch, tdelete, twalk: . . . . . tsearch(3C)
  - tsort: topological sort. . . . . tsort(1)
  - ttt, cubic: tic-tac-toe. . . . . ttt(6)
  - tty: get the terminal's name.
  - TTY-37 type-box. greek: . . . . . greek(5)
  - ttynname, isatty: find name of . . . . . ttynname(3C)
  - ttyslot: find the slot in the . . . . . ttyslot(3C)
  - twalk: manage binary search . . . . . tsearch(3C)
  - type conversion. /dcmplx, . . . . . ftype(3F)
  - type. . . . . file(1)
  - type. /vax: provide truth . . . . . machid(1)
  - type-box. greek: graphics . . . . . greek(5)
  - types: primitive system data . . . . . types(5)
  - types. . . . . types(5)
  - typeset documents, view . . . . . mmt(1)
  - typeset text. . . . . troff(1)
  - typesetting view graphs and/ . . . . . mv(5)
  - tzset: convert date and time/ . . . . . ctime(3C)
  - u3b, u3b5, vax: provide truth . . . . . machid(1)
  - u3b5, vax: provide truth value . . . . . machid(1)
  - UID. . . . . getpw(3C)
  - ulimit: get and set user . . . . . ulimit(2)
  - umask: set and get file . . . . . umask(2)
  - umask: set file-creation mode . . . . . umask(1)
  - umount: unmount a file system. . . . . umount(2)
  - uname: get name of current . . . . . uname(2)
  - uname: print name of current . . . . . uname(1)
  - undo a previous get of an SCCS . . . . . unget(1)
  - unget: undo a previous get of . . . . . unget(1)
  - ungetc: push character back . . . . . ungetc(3S)
  - uniform random-number/ . . . . . rand(3F)
  - uniformly distributed/ . . . . . drand48(3C)
  - uniq: report repeated lines in . . . . . uniq(1)
  - unique file name. . . . . mktemp(3C)
  - units: conversion program. . . . . units(1)
  - UNIX-to-UNIX file copy. . . . . uuto(1C)
  - unlink: remove directory . . . . . unlink(2)
  - unmount a file system. . . . . umount(2)
  - unpack: compress and expand . . . . . pack(1)

times of a file. touch: update access and modification . . . . touch(1)  
of programs. make: maintain, update, and regenerate groups . . . . make(1)  
search: linear search and update. . . . lsearch(3C)  
sync: update super-block. . . . sync(2)  
sync: update the super block. . . . sync(1)  
du: summarize disk usage. . . . du(1)  
stat: statistical network useful with graphical/ . . . . stat(1G)  
id: print user and group IDs and names. . . . id(1)  
setuid, setgid: set user and group IDs. . . . setuid(2)  
character login name of the user. cuserid: get . . . . cuserid(3S)  
/getgid, getegid: get real user, effective user, real/  
environ: user environment. . . . environ(5)  
gcosmail: send mail to HIS user. . . . gcosmail(1C)  
ulimit: get and set user limits. . . . ulimit(2)  
logname: return login name of user. . . . logname(3X)  
/get real user, effective user, real group, and/  
become super-user or another user. su: . . . . su(1)  
the utmp file of the current user. /find the slot in . . . . ttyslot(3C)  
write: write to another user. . . . write(1)  
mail, rmail: send mail to users or read mail. . . . mail(1)  
statistics. ustat: get file system . . . . ustat(2)  
gutil: graphical utilities. . . . gutil(1G)  
modification times. utime: set file access and . . . . utime(2)  
utmp, wtmp: utmp and wtmp entry formats. . . . utmp(4)  
endument, utmpname: access utmp file entry. /setuent, . . . . setut(3C)  
ttyslot: find the slot in the utmp file of the current user. . . . ttyslot(3C)  
entry formats. utmp, wtmp: utmp and wtmp . . . . utmp(4)  
/pututline, setuent, endument, utmpname: access utmp file/  
control. uustat: uucp status inquiry and job . . . . uustat(1C)  
unix copy. uucp, uulog, uuname: unix to . . . . uucp(1C)  
copy. uucp, uulog, uuname: unix to unix . . . . uucp(1C)  
uucp, uulog, uuto, and job control. uuname: unix to unix copy. . . . uucp(1C)  
UNIX-to-UNIX file copy. uupick: public UNIX-to-UNIX . . . . uuto(1C)  
execution. uustat: uucp status inquiry . . . . uustat(1C)  
uuto, uupick: public uuto(1C)  
uux: unix to unix command . . . . uux(1C)  
val: validate SCCS file. . . . val(1)  
validate SCCS file. . . . val(1)  
/u3b, u3b5, vax: provide truth value about your processor/ . . . . machid(1)  
abs: return integer absolute value. . . . abs(3C)  
cabs, zabs: Fortran absolute value. abs, iabs, dabs, . . . . abs(3F)  
getenv: return value for environment name. . . . getenv(3C)  
ceiling, remainder, absolute value functions. /fabs: floor, . . . . floor(3M)  
true, false: provide truth values. . . . true(1)  
return Fortran environment variable. getenv: . . . . getenv(3F)  
your/ pdp11, u3b, u3b5, vax: provide truth value about . . . . machid(1)  
option letter from argument vc: version control. . . . vc(1)  
assert: vector. getopt: get . . . . getopt(3C)  
vpr: Versatec printer spooler. . . . vpr(1)  
vc: version control. . . . vc(1)  
get: get a version of an SCCS file. . . . get(1)  
scsdiff: compare two versions of an SCCS file. . . . scsdiff(1)  
se: screen editor for video terminals. . . . se(1)  
mmt, mvt: typeset documents, view graphs, and slides. . . . mmt(1)  
macro package for typesetting view graphs and slides. /troff . . . . mv(5)  
file system: format of system volume. . . . fs(4)  
process. vpr: Versatec printer spooler. . . . vpr(1)  
or terminate. wait: wait: await completion of . . . . wait(1)  
to stop or terminate. wait: wait for child process to stop . . . . wait(2)  
wait: wait for child process . . . . wait(2)  
ftw: walk a file tree. . . . ftw(3C)  
wc: word count. . . . wc(1)  
what: identify SCCS files. . . . what(1)

- XMOU -

signal. signal: specify what to do upon receipt of a . . . . . signal(2)  
                   who: who is on the system. . . . . who(1)  
                   who: who is on the system. . . . . who(1)  
                   cd: change working directory. . . . . cd(1)  
                   chdir: change working directory. . . . . chdir(2)  
 get path-name of current working directory. getcwd: . . . . . getcwd(3C)  
                   pwd: working directory name. . . . . pwd(1)  
                   write: write on a file. . . . . write(2)  
                   putpwent: write password file entry. . . . . putpwent(3C)  
                   write: write to another user. . . . . write(1)  
                   write: write on a file. . . . . write(2)  
                   write: write to another user. . . . . write(1)  
 open: open for reading or writing. . . . . open(2)  
       utmp, wtmp: utmp and wtmp entry formats. . . . . utmp(4)  
           formats. utmp, wtmp: utmp and wtmp entry . . . . . utmp(4)  
           hunt-the-wumpus. wump: the game of . . . . . wump(6)  
       install a BX.25 link. x25alnk, x25ilnk: attach or . . . . . x25alnk(3C)  
           link. x25clnk: change over a BX.25 . . . . . x25clnk(3C)  
           BX.25 link. x25hlnk, x25dlnk: halt or detach a . . . . . x25hlnk(3C)  
           detach a BX.25 link. x25ilnk, x25dlnk: halt or . . . . . x25hlnk(3C)  
           BX.25 link. x25alnk, x25ilnk: attach or install a . . . . . x25alnk(3C)  
           remove a PVC on a link. x25ipvc, x25rpvc: install or . . . . . x25ipvc(3C)  
           PVC on a link. x25ipvc, x25rpvc: install or remove a . . . . . x25ipvc(3C)  
 list(s) and execute command. xargs: construct argument . . . . . xargs(1)  
       Fortran bitwise/ and, or, xor, not, lshift, rshift: . . . . . bool(3F)  
           j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)  
           j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)  
           compiler-compiler. yacc: yet another . . . . . yacc(1)  
           j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)  
           abs, iabs, dabs, cabs, zabs: Fortran absolute value. . . . . abs(3F)

INDEX





**NAME**

intro — introduction to commands and application programs

**DESCRIPTION**

This section describes, in alphabetical order, publicly-accessible commands. Certain distinctions of purpose are made in the headings:

- (1) Commands of general utility.
- (1C) Commands for communication with other systems.
- (1G) Commands used primarily for graphics and computer-aided design.

**COMMAND SYNTAX**

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

*name* [*option(s)*] [*cmdarg(s)*]

where:

<i>name</i>	The name of an executable file.
<i>option</i>	— <i>noargletter(s)</i> or, — <i>argletter</i> <> <i>optarg</i> where <> is optional white space.
<i>noargletter</i>	A single letter representing an option without an argument.
<i>argletter</i>	A single letter representing an option requiring an argument.
<i>optarg</i>	Argument (character string) satisfying preceding <i>argletter</i> .
<i>cmdarg</i>	Path name (or other command argument) <i>not</i> beginning with — or, — by itself indicating the standard input.

**SEE ALSO**

getopt(1), getopt(3C).  
Section 6 of this volume for computer games.  
*How to Get Started*, at the front of this volume.

**DIAGNOSTICS**

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of “normal” termination) one supplied by the program (see *wait(2)* and *exit(2)*). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero 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.

**BUGS**

Regretfully, many commands do not adhere to the aforementioned syntax.

**NAME**

300, 300s — handle special functions of DASI 300 and 300s terminals

**SYNOPSIS**

**300** [ +12 ] [ -n ] [ -dt,l,c ]

**300s** [ +12 ] [ -n ] [ -dt,l,c ]

**DESCRIPTION**

**300** supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; **300s** performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It also reduces printing time 5 to 70%. **300** can be used to print equations neatly, in the sequence:

```
neqn file ... | nroff | 300
```

**WARNING:** if your terminal has a PLOT switch, make sure it is turned *on* before **300** is used.

The behavior of **300** can be modified by the optional flag arguments to handle 12-pitch text, fractional line spacings, messages, and delays.

**+12** permits use of 12-pitch, 6 lines/inch text. DASI 300 terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch. To obtain the 12-pitch, 6 lines per inch combination, the user should turn the PITCH switch to 12, and use the **+12** option.

**-n** controls the size of half-line spacing. A half-line is, by default, equal to 4 vertical plot increments. Because each increment equals 1/48 of an inch, a 10-pitch line-feed requires 8 increments, while a 12-pitch line-feed needs only 6. The first digit of *n* overrides the default value, thus allowing for individual taste in the appearance of subscripts and superscripts. For example, *nroff* half-lines could be made to act as quarter-lines by using **-2**. The user could also obtain appropriate half-lines for 12-pitch, 8 lines/inch mode by using the option **-3** alone, having set the PITCH switch to 12-pitch.

**-dt,l,c** controls delay factors. The default setting is **-d3,90,30**. DASI 300 terminals sometimes produce peculiar output when faced with very long lines, too many tab characters, or long strings of blankless, non-identical characters. One null (delay) character is inserted in a line for every set of *t* tabs, and for every contiguous string of *c* non-blank, non-tab characters. If a line is longer than *l* bytes,  $1 + (\text{total length})/20$  nulls are inserted at the end of that line. Items can be omitted from the end of the list, implying use of the default values. Also, a value of zero for *t* (*c*) results in two null bytes per tab (character). The former may be needed for C programs, the latter for files like */etc/passwd*. Because terminal behavior varies according to the specific characters printed and the load on a system, the user may have to experiment with these values to get correct output. The **-d** option exists only as a last resort for those few cases that do not otherwise print properly. For example, the file */etc/passwd* may be printed using **-d3,30,5**. The value **-d0,1** is a good one to use for C programs that have many levels of indentation.

Note that the delay control interacts heavily with the prevailing carriage return and line-feed delays. The *stty*(1) modes **n10 cr2** or **n10 cr3** are recommended for most uses.

*300* can be used with the *nroff* -s flag or .rd requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the following sequences are equivalent:

```
nroff -T300 files ... and nroff files ... | 300
nroff -T300-12 files ... and nroff files ... | 300 +12
```

The use of *300* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *300* may produce better-aligned output.

The *neqn* names of, and resulting output for, the Greek and special characters supported by *300* are shown in *greek*(5).

#### SEE ALSO

*450*(1), *eqn*(1), *graph*(1G), *mesg*(1), *nroff*(1), *stty*(1), *tabs*(1), *tbl*(1), *tplot*(1G), *greek*(5).

#### BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

**NAME**

4014 — paginator for the Tektronix 4014 terminal

**SYNOPSIS**

**4014** [ *-t* ] [ *-n* ] [ *-cN* ] [ *-pL* ] [ *file* ]

**DESCRIPTION**

The output of *4014* is intended for a Tektronix 4014 terminal; *4014* arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight-space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. TELETYPE® Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page, *4014* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command *!cmd* will send the *cmd* to the shell.

The command line options are:

- t* Don't wait between pages (useful for directing output into a file).
- n* Start printing at the current cursor position and never erase the screen.
- cN* Divide the screen into *N* columns and wait after the last column.
- pL* Set page length to *L*; *L* accepts the scale factors *i* (inches) and *l* (lines); default is lines.

**SEE ALSO**

*pr(1)*, *tc(1)*, *troff(1)*.

**NAME**

450 — handle special functions of the DASI 450 terminal

**SYNOPSIS**

450

**DESCRIPTION**

450 supports special functions of, and optimizes the use of, the DASI 450 terminal, or any terminal that is functionally identical, such as the DIABLO 1620 or XEROX 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as 300(1). 450 can be used to print equations neatly, in the sequence:

```
neqn file ... | nroff | 450
```

**WARNING:** make sure that the PLOT switch on your terminal is ON before 450 is used. The SPACING switch should be put in the desired position (either 10- or 12-pitch). In either case, vertical spacing is 6 lines/inch, unless dynamically changed to 8 lines per inch by an appropriate escape sequence.

450 can be used with the *nroff* -s flag or .rd requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the use of 450 can be eliminated in favor of one of the following:

```
nroff -T450 files ...
```

or

```
nroff -T450-12 files ...
```

The use of 450 can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of 450 may produce better-aligned output.

The *neqn* names of, and resulting output for, the Greek and special characters supported by 450 are shown in *greek(5)*.

**SEE ALSO**

300(1), eqn(1), graph(1G), msg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G), greek(5).

**BUGS**

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

## NAME

acctcom — search and print process accounting file(s)

## SYNOPSIS

**acctcom** [[options][file]] . . .

## DESCRIPTION

*Acctcom* reads *file*, the standard input, or `/usr/adm/pacct`, in the form described by *acct*(4) 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*) and STAT (the system exit status).

The command name is prepended with a # if it 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 *files* are specified, and if the standard input is associated with a terminal or `/dev/null` (as is the case when using `&` in the shell), `/usr/adm/pacct` is read, otherwise the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward, i.e., in chronological order by process completion time. The file `/usr/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 `/usr/adm/pacct?`. The *options* are:

- b Read backwards, showing latest commands first.
- f Print the *fork/exec* flag and system exit status columns in the output.
- 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).
- 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.
- l *line* Show only processes belonging to terminal `/dev/line`.
- 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.
- g *group* Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.
- d *mm/dd* Any *time* arguments following this flag are assumed to occur on the given month *mm* and the day *dd* rather than during last 24 hours. This is needed for looking at old files.
- s *time* Select processes existing at or after *time*, given in the format *hr* [:*min* [:*sec*]].
- e *time* Select processes existing at or before *time*.
- S *time* Select processes starting at or after *time*.
- E *time* Select processes ending at or before *time*.
- n *pattern* Show only commands matching *pattern* that may be a regular expression as in *ed*(1) except that + means one or more occurrences.

- o *ofile* Copy selected process records in the input data format to *ofile*; suppress standard output printing.
- H *factor* Show only processes that exceed *factor*, where *factor* is the "hog factor" as explained in option -h above.
- O *sec* Show only processes with CPU system time exceeding *sec* seconds.
- C *sec* Show only processes with total CPU time, system plus user, exceeding *sec* seconds.

Listing options together has the effect of a logical *and*.

#### FILES

/etc/passwd  
 /usr/adm/pacct  
 /etc/group

#### SEE ALSO

ps(1), su(1), acct(2), acct(4), utmp(4).  
 acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M),  
 acctsh(1M), fwtmp(1M), runacct(1M) in the *UNIX System Administrator's Manual*.

#### BUGS

*Acctcom* only reports on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time and option -d is not used, then *time* is interpreted as occurring on the previous day.

1



## NAME

**adb** — absolute debugger

## SYNOPSIS

**adb** [-w] [ objfil [ corfil ] ]

## DESCRIPTION

*Adb* is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

*Objfil* is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of *adb* cannot be used although the file can still be examined. The default for *objfil* is **a.out**. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is **core**.

Requests to *adb* are read from the standard input and responses are to the standard output. If the **-w** flag is present then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using *adb*. *Adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

In general requests to *adb* are of the form

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

If *address* is present then *dot* is set to *address*. Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see *ADDRESSES*.

## 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.

*integer* An octal number if *integer* begins with a 0; a hexadecimal number if preceded by #; otherwise a decimal number.

*integer.fraction*

A 32 bit floating point number.

'*cccc*' The ASCII value of up to 4 characters. \ may be used to escape a '.

< *name*

The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables (see *VARIABLES*) named by single letters or digits. If *name* is a register name then the value of the register is obtained from the system header in *corfil*. The register names are **r0 ... r5 sp pc ps**.

*symbol* A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. The value of the *symbol* is taken from the symbol table in *objfil*. An initial **\_** or **~** will be prefixed to *symbol* if needed.

**\_** *symbol*

In C, the "true name" of an external symbol begins with **\_**. It may

be necessary to utter this name to distinguish it from internal or hidden variables of a program.

*routine.name*

The address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols*. If *name* is omitted the value is the address of the most recently activated C stack frame corresponding to *routine*.

(*exp*) The value of the expression *exp*.

Monadic operators:

\**exp* The contents of the location addressed by *exp* in *corfil*.

@*exp* The contents of the location addressed by *exp* in *objfil*.

-*exp* Integer negation.

~*exp* Bitwise complement.

Dyadic operators are left associative and are less binding than monadic operators.

*e1 + e2* Integer addition.

*e1 - e2* Integer subtraction.

*e1 \* e2* Integer multiplication.

*e1 % e2* Integer division.

*e1 & e2* Bitwise conjunction.

*e1 | e2* Bitwise disjunction.

*e1 # e2* *E1* rounded up to the next multiple of *e2*.

## COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands ? and / may be followed by \*; see *ADDRESSES* for further details.)

?*f* Locations starting at *address* in *objfil* are printed according to the format *f*. *dot* is incremented by the sum of the increments for each format letter (q.v.).

/*f* Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is incremented as for ?.

=*f* The value of *address* itself is printed in the styles indicated by the format *f*. (For i format ? is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format *dot* is incremented by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows:

- o 2 Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
- 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.
- u 2 Print as an unsigned decimal number.
- U 4 Print long unsigned decimal.
- f 4 Print the 32 bit value as a floating point number.
- F 8 Print double floating point.
- b 1 Print the addressed byte in octal.
- c 1 Print the addressed character.
- C 1 Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.
- s n Print the addressed characters until a zero character is reached.
- S n Print a string using the @ escape convention. n is the length of the string including its zero terminator.
- Y 4 Print 4 bytes in date format (see *ctime(3C)*).
- i n Print as PDP-11 instructions. n is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
- a 0 Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.
  - / local or global data symbol
  - ? local or global text symbol
  - = local or global absolute symbol
- p 2 Print the addressed value in symbolic form using the same rules for symbol lookup as a.
- t 0 When preceded by an integer tabs to the next appropriate tab stop. For example, 8t moves to the next 8-space tab stop.
- r 0 Print a space.
- n 0 Print a new-line.
- \*...\* 0 Print the enclosed string.
  - Dot* is decremented by the current increment. Nothing is printed.
  - + *Dot* is incremented by 1. Nothing is printed.
  - *Dot* is decremented by 1. Nothing is printed.

**new-line**

Repeat the previous command with a *count* of 1.

**[?/] value mask**

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If L is used then the match is for 4 bytes at a time instead of 2. If no match is found then *dot* is unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted then -1 is used.

**[?/]w value ...**

Write the 2-byte *value* into the addressed location. If the command is W, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

**[?/]m bl el fl[?/]**

New values for (*bl*, *el*, *fl*) are recorded. If less than three expressions are given then the remaining map parameters are left

unchanged. If the ? or / is followed by \* then the second segment (*b2, e2, f2*) of the mapping is changed. If the list is terminated by ? or / then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (So that, for example, /m? will cause / to refer to *objfil*.)

>name

*Dot* is assigned to the variable or register named.

! A shell is called to read the rest of the line following !.

\$modifier

Miscellaneous commands. The available *modifiers* are:

- <*f* Read commands from the file *f* and return.
- >*f* Send output to the file *f*, which is created if it does not exist.
- r Print the general registers and the instruction addressed by *pc*. *Dot* is set to *pc*.
- f Print the floating registers in single or double length. If the floating point status of *ps* is set to double (0200 bit) then double length is used anyway.
- b Print all breakpoints and their associated counts and commands.
- a ALGOL 68 stack backtrace. If *address* is given then it is taken to be the address of the current frame (instead of *r4*). If *count* is given then only the first *count* frames are printed.
- c C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of *r5*). If C is used then the names and (16 bit) values of all automatic and static variables are printed for each active function. If *count* is given then only the first *count* frames are printed.
- e The names and values of external variables are printed.
- 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.
- d Reset integer input as described in *EXPRESSIONS*.
- q Exit from *adb*.
- v Print all non zero variables in octal.
- m Print the address map.

:modifier

Manage a subprocess. Available modifiers are:

- bc Set breakpoint at *address*. The breakpoint is executed *count*-1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command sets *dot* to zero then the breakpoint causes a stop.
- d Delete breakpoint at *address*.
- r Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on

entry to the subprocess.

- cs** The subprocess is continued with signal *s* (see *signal(2)*). If *address* is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for *r*.
- ss** As for *c* except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfil* is run as a subprocess as for *r*. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.
- k** The current subprocess, if any, is terminated.

## VARIABLES

*Adb* provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

- 0** The last value printed.  
**1** The last offset part of an instruction source.  
**2** The previous value of variable 1.

On entry the following are set from the system header in the *corfil*. If *corfil* does not appear to be a core file then these values are set from *objfil*.

- b** The base address of the data segment.  
**d** The data segment size.  
**e** The entry point.  
**m** The "magic" number (0405, 0407, 0410 or 0411).  
**s** The stack segment size.  
**t** The text segment size.

## ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1*, *e1*, *f1*) and (*b2*, *e2*, *f2*) and the *file address* corresponding to a written *address* is calculated as follows:

$$b1 \leq \text{address} < e1 \Rightarrow \text{file address} = \text{address} + f1 - b1$$

otherwise

$$b2 \leq \text{address} < e2 \Rightarrow \text{file address} = \text{address} + f2 - b2,$$

otherwise, the requested *address* is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a ? or / is followed by an \* then only the second triple is used.

The initial setting of both mappings is suitable for normal *a.out* and *core* files. If either file is not of the kind expected then, for that file, *b1* is set to 0, *e1* is set to the maximum file size and *f1* is set to 0; in this way the whole file can be examined with no address translation.

In order for *adb* to be used on large files all appropriate values are kept as signed 32 bit integers.

## FILES

/dev/mem  
 /dev/swap  
 a.out  
 core

**SEE ALSO**

ptrace(2), a.out(4), core(4).

**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**

A breakpoint set at the entry point is not effective on initial entry to the program.

When single stepping, system calls do not count as an executed instruction. Local variables whose names are the same as an external variable may foul up the accessing of the external.

## NAME

`admin` — create and administer SCCS files

## SYNOPSIS

```
admin [-n] [-i[name]] [-rrel] [-t[name]] [-fflag[flag-val]]
[-dflag[flag-val]] [-alogin] [-elogin] [-m[mrlist]] [-y[comment]]
[-h] [-z] files
```

## DESCRIPTION

*Admin* is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which may appear in any order, consist of keyletter arguments, which begin with `-`, and named files (note that SCCS file names must begin with the characters `s.`). If a named file doesn't exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, *admin* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with `s.`) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

- `-n` This keyletter indicates that a new SCCS file is to be created.
- `-i[name]` The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see `-r` keyletter for delta numbering scheme). If the `i` keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an *admin* command on which the `i` keyletter is supplied. Using a single *admin* to create two or more SCCS files require that they be created empty (no `-i` keyletter). Note that the `-i` keyletter implies the `-n` keyletter.
- `-rrel` The *release* into which the initial delta is inserted. This keyletter may be used only if the `-i` keyletter is also used. If the `-r` keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).
- `-t[name]` The *name* of a file from which descriptive text for the SCCS file is to be taken. If the `-t` keyletter is used and *admin* is creating a new SCCS file (the `-n` and/or `-i` keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a `-t` keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a `-t` keyletter with a file

- name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.
- f***flag* This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several *f* keyletters may be supplied on a single *admin* command line. The allowable *flags* and their values are:
- b** Allows use of the **-b** keyletter on a *get*(1) command to create branch deltas.
- c***ceil* The highest release (i.e., "ceiling"), a number less than or equal to 9999, which may be retrieved by a *get*(1) command for editing. The default value for an unspecified *c* flag is 9999.
- f***floor* The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a *get*(1) command for editing. The default value for an unspecified *f* flag is 1.
- d***SID* The default delta number (SID) to be used by a *get*(1) command.
- i** Causes the "No id keywords (ge6)" message issued by *get*(1) or *delta*(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*(1)) are found in the text retrieved or stored in the SCCS file.
- j** Allows concurrent *get*(1) commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.
- l***list* A *list* of releases to which deltas can no longer be made (*get* -e against one of these "locked" releases fails). The *list* has the following syntax:
- ```
<list> ::= <range> | <list> , <range>
<range> ::= RELEASE NUMBER | a
```
- The character **a** in the *list* is equivalent to specifying *all releases* for the named SCCS file.
- n** Causes *delta*(1) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file preventing branch deltas from being created from them in the future.
- q***text* User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by *get*(1).
- m***mod* Module name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by *get*(1). If the **m** flag is not specified, the value assigned is the name of the SCCS file with the



leading s. removed.

- ttype** Type of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by *get*(1).
- v[pgm]** Causes *delta*(1) to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validity checking program (see *delta*(1)). (If this flag is set when creating an SCCS file, the **m** keyletter must also be used even if its value is null).
- dflag** Causes removal (deletion) of the specified *flag* from an SCCS file. The **-d** keyletter may be specified only when processing existing SCCS files. Several **-d** keyletters may be supplied on a single *admin* command. See the **-f** keyletter for allowable *flag* names.
- llist** A *list* of releases to be "unlocked". See the **-f** keyletter for a description of the **l** flag and the syntax of a *list*.
- alogin** A *login* name, or numerical UNIX group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several a keyletters may be used on a single *admin* command line. As many *logins*, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas.
- elogin** A *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several e keyletters may be used on a single *admin* command line.
- y[comment]** The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(1). Omission of the **-y** keyletter results in a default comment line being inserted in the form:  
date and time created YY/MM/DD HH:MM:SS by *login*  
The **-y** keyletter is valid only if the **-i** and/or **-n** keyletters are specified (i.e., a new SCCS file is being created).
- m[mrlist]** The list of Modification Requests (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*(1). The **v** flag must be set and the MR numbers are validated if the **v** flag has a value (the name of an MR number validation program). Diagnostics will occur if the **v** flag is not set or MR validation fails.
- h** Causes *admin* to check the structure of the SCCS file (see *scsfile*(5)), and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum

that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

**-z** The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **-h**, above).

Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

## FILES

The last component of all SCCS file names must be of the form *s.file-name*. New SCCS files are given mode 444 (see *chmod(1)*). Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file, called *x.file-name*, (see *get(1)*), created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by 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 allowing use of *ed(1)*. *Care must be taken!* 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 the SCCS file is valid.

*Admin* also makes use of a transient lock file (called *z.file-name*), which is used to prevent simultaneous updates to the SCCS file by different users. See *get(1)* for further information.

## SEE ALSO

*delta(1)*, *ed(1)*, *get(1)*, *help(1)*, *prs(1)*, *what(1)*, *sccsfile(4)*.

*Source Code Control System User's Guide* in the *UNIX System User's Guide*.

## DIAGNOSTICS

Use *help(1)* for explanations.

## NAME

*ar* — archive and library maintainer for portable archives

## SYNOPSIS

*ar* key [ *posname* ] *afile* name ...

## DESCRIPTION

*Ar* maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose.

When *ar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in *ar(4)*. The archive symbol table (described in *ar(4)*) is used by the link editor (*ld(1)*) to effect multiple passes over libraries of object files in an efficient manner. Whenever the *ar(1)* command is used to create or update the contents of an archive, the symbol table is rebuilt. The symbol table can be forced to be rebuilt by the *s* option described below.

*Key* is one character from the set *drqtpmx*, optionally concatenated with one or more of *vuaijbcls*. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

- d** Delete the named files from the archive file.
- r** Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.
- q** Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.
- t** Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.
- p** Print the named files in the archive.
- m** Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.
- x** Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.
- v** Verbose. Under the verbose option, *ar* gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the files. When used with **x**, it precedes each file with a name.
- c** Create. Normally *ar* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.
- l** Local. Normally *ar* places its temporary files in the directory */tmp*. This option causes them to be placed in the local directory.

- s Symbol table creation. Force the regeneration of the archive symbol table even if *ar*(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the *strip*(1) command has been used on the archive.

**FILES**

/tmp/ar\*            temporaries

**SEE ALSO**

arcv(1), ld(1), lorder(1), a.out(4), ar(4).

**BUGS**

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

## NAME

*ar* — archive and library maintainer

## SYNOPSIS

*ar* key [ *posname* ] *afile* name ...

## DESCRIPTION

*Ar* maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose.

When *ar* creates an archive, it always creates the header in the format of the local system. A conversion program exists to convert PDP-11 archives to pre-UNIX 5.0 VAX-11/780 archive format (see *arcv*(1)). Another conversion program, *convert*(1), exists on the VAX and 3B20S to convert archives from the pre-UNIX 5.0 format to the "common" archive format described in *ar*(4). Individual files are inserted without conversion into the archive file.

*Key* is one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcl**. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

- d** Delete the named files from the archive file.
- r** Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.
- q** Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.
- t** Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.
- p** Print the named files in the archive.
- m** Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.
- x** Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.
- v** Verbose. Under the verbose option, *ar* gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the files. When used with **x**, it precedes each file with a name.
- c** Create. Normally *ar* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.
- l** Local. Normally *ar* places its temporary files in the directory **/tmp**. This option causes them to be placed in the local directory.

**FILES**

/tmp/v\*            temporaries

**SEE ALSO**

arcv(1), ld(1), lorder(1), ar(4).

**BUGS**

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



**NAME**

arcv — convert archive files from PDP-11 to common archive format

**SYNOPSIS**

arcv infile outfile

**DESCRIPTION**

*Arvcv* converts source archive files from the PDP-11 format to the UNIX 5.0 portable archive format. The input archive file *infile* is converted to an equivalent output archive file *outfile*. Note that there is no conversion of the members of the input archive file.

**FILES**

/tmp/arcv\*

**SEE ALSO**

ar(1), convert(1), ar(4).

**NAME**

as — common assembler

**SYNOPSIS**

as [-o *objfile*] [-n] [-m] [-R] [-r] [-*bwl*] [-V] file-name

**DESCRIPTION**

The *as* command assembles the named file. The following flags may be specified in any order:

- o *objfile* Output of assembly is put in *objfile*. By default, the output file name is formed by removing the *.s* suffix, if there is one, from the input file name and appending a *.o* suffix.
- n Turns off long/short address optimization. By default, address optimization takes place.
- m Runs the *m4* macro pre-processor on the input to the assembler.
- R Instructs the assembler to delete (unlink) the input file after assembly is completed. This option is off by default.
- r For the VAX version of the common assembler only. This option instructs the assembler to place all assembled data (normally placed in the *.data* section) into the *.text* section. This option effectively disables the *.data* pseudo operation. This option is off by default.
- bwl* For the VAX version of the common assembler only. This option instructs the assembler to create byte (*b*), halfword (*w*) or long (*l*) displacements for undefined symbols. The default value for this option is long (*l*) displacements.
- V Causes the version number of the assembler being run to be written on standard error.

**FILES**

/usr/tmp/as[1-6]XXXXXX temporary files

**SEE ALSO**

ld(1), m4(1), nm(1), strip(1), a.out(4).

**DIAGNOSTICS**

If the input file cannot be read, the assembly will terminate with the message "Unable to open input file". If assembly errors are detected the following information is written to standard error: the input file name, line number where the error occurred in the assembly code, a (hopefully) descriptive message of the problem, and, if the input file was produced by the C compiler (see *cc*(1)) the line number in the C program that generated the erroneous code.

**CAVEATS**

Those running the assembler explicitly should take note of some possible pitfalls:

- If the *-m* (*m4* macro pre-processor invocation) option is used, keywords for *m4* (see *m4*(1)) cannot be used as symbols (variables, functions, labels) in the input file since *m4* cannot determine which are assembler symbols and which are real *m4* macros.

**BUGS**

The *.align* assembler directive is not guaranteed to work in the *.text* section when optimization is performed.

Arithmetic expressions may only have one forward referenced symbol per expression.



**NAME**

as — assembler for PDP-11

**SYNOPSIS**

as [ - ] [ -o objfile ] file ...

**DESCRIPTION**

As assembles the concatenation of the named files. If the optional first argument — is used, all undefined symbols in the assembly are treated as global.

The output of the assembly is left on the file *objfile*; if that is omitted, *a.out* is used. It is executable if no errors occurred during the assembly, and if there were no unresolved external references.

**FILES**

|                |                         |
|----------------|-------------------------|
| /lib/as2       | pass 2 of the assembler |
| /tmp/atm[1-3]? | temporary               |
| a.out          | object                  |

**SEE ALSO**

adb(1), ld(1), nm(1), a.out(4).  
*UNIX Assembler Manual* by D. M. Ritchie.

**DIAGNOSTICS**

If the name chosen for the output file is of the form *\*?.[cs]*, the assembler issues an appropriate complaint and quits. When an input file cannot be read, its name followed by a question mark is typed and assembly ceases. When syntactic or semantic errors occur, a single-character diagnostic is typed out together with the line number and the file name in which it occurred. Errors in pass 1 cause cancellation of pass 2. The possible errors are:

|   |                                         |
|---|-----------------------------------------|
| ) | Parentheses error                       |
| ] | Parentheses error                       |
| < | String not terminated properly          |
| * | Indirection used illegally              |
| . | Illegal assignment to .                 |
| a | Error in address                        |
| b | Branch instruction is odd or too remote |
| e | Error in expression                     |
| f | Error in local (f or b) type symbol     |
| g | Garbage (unknown) character             |
| i | End of file inside an <i>.if</i>        |
| m | Multiply-defined symbol as label        |
| o | Word quantity assembled at odd address  |
| p | . different in pass 1 and 2             |
| r | Relocation error                        |
| u | Undefined symbol                        |
| x | Syntax error                            |

**BUGS**

Syntax errors can cause incorrect line numbers in subsequent diagnostics.

**NAME**

*asa* — interpret ASA carriage control characters

**SYNOPSIS**

*asa* [files]

**DESCRIPTION**

*Asa* interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments or the stand:d input if no file names are supplied. The first character of each line is assumed to be a control character; their meanings are:

' ' (blank) single new line before printing  
0 double new line before printing  
1 new page before printing  
+ overprint previous line.

Lines beginning with other than the above characters are treated as if they began with ' '. The first character of a line is *not* printed. If any such lines appear, an appropriate diagnostic will appear on standard error. This program forces the first line of each input file to start on a new page.

To correctly view the output of FORTRAN programs which use ASA carriage control characters, *asa* could be used as a filter thusly:

```
a.out |asa |lpr
```

and the output, properly formatted and pagenated, would be directed to the line printer. FORTRAN output sent to a file could be viewed by:

```
asa file
```

**SEE ALSO**

efl(1), f77(1), fsplit(1), ratfor(1).

1

**NAME**

**awk** — pattern scanning and processing language

**SYNOPSIS**

**awk** [ *-F*c ] [ *prog* ] [ *parameters* ] [ *files* ]

**DESCRIPTION**

*Awk* scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as *-f file*. The *prog* string should be enclosed in single quotes (') to protect it from the shell.

*Parameters*, in the form *x=... y=... etc.*, may be passed to *awk*.

Files are read in order; if there are no files, the standard input is read. The file name *-* means the standard input. Each 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 made up of fields separated by white space. (This default can be changed by using *FS*, see below). The fields are denoted *\$1*, *\$2*, ...; *\$0* refers to the entire line.

A pattern-action statement has the form:

```
pattern { action }
```

A missing action means print the line; a missing pattern always matches. 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 ]
printf format [ , expression-list ] [ >expression ]
next # skip remaining patterns on this input line
exit # skip the rest of the input
```

Statements are terminated by semicolons, new-lines, 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. Variables 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; 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 *>expr* 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 (see *printf(3S)*).

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*, *sqr*, and *int*. The last truncates its argument to an integer;

*substr*(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf*(*fmt*, *expr*, *expr*, ...) formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

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*(1)). 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 ~ (for *contains*) or !~ (for *does not contain*). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

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

```
BEGIN { FS = c }
```

or by using the -Fc 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 new-line); 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 }
```

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }  
{ print }
```

command line: `awk -f program n==5 input`

**SEE ALSO**

`grep(1)`, `lex(1)`, `sed(1)`.

*Awk—A Pattern Scanning and Processing Language* by A. V. Aho, B. W. Kernighan, and P. J. Weinberger.

**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.

**NAME**

banner — make posters

**SYNOPSIS**

banner strings

**DESCRIPTION**

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

**SEE ALSO**

echo(1).

**NAME**

**basename**, **dirname** — deliver portions of path names

**SYNOPSIS**

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

**DESCRIPTION**

*Basename* deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside substitution marks (^~) within shell procedures.

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

**EXAMPLES**

The following example, invoked with the argument `/usr/src/cmd/cat.c`, compiles the named file and moves the output to a file named `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).

**BUGS**

The *basename* of / is null and is considered an error.

**NAME**

*bc* — arbitrary-precision arithmetic language

**SYNOPSIS**

*bc* [ *-c* ] [ *-l* ] [ *file ...* ]

**DESCRIPTION**

*Bc* is an interactive processor for a language that resembles *C* but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The *-l* argument stands for the name of an arbitrary precision math library. The syntax for *bc* programs is as follows; *L* means letter *a-z*, *E* means expression, *S* means statement.

**Comments**

are enclosed in */\** and *\*/*.

**Names**

simple variables: *L*

array elements: *L [ E ]*

The words “ibase”, “obase”, and “scale”

**Other operands**

arbitrarily long numbers with optional sign and decimal point.

( *E* )

sqrt ( *E* )

length ( *E* )      number of significant decimal digits

scale ( *E* )      number of digits right of decimal point

*L* ( *E* , ... , *E* )

**Operators**

+ - \* / % ^ (% is remainder; ^ is power)

++ --      (prefix and postfix; apply to names)

== <= >= != < >

- = + - = - \* = / = % = ^

**Statements**

*E*

{ *S* ; ... ; *S* }

if ( *E* ) *S*

while ( *E* ) *S*

for ( *E* ; *E* ; *E* ) *S*

null statement

break

quit

**Function definitions**

define *L* ( *L* , ... , *L* ) {

    auto *L* , ... , *L*

*S* ; ... *S*

    return ( *E* )

}

**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 new-lines 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.

*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.

#### EXAMPLE

```
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
    }
}
```

defines a function to compute an approximate value of the exponential function and

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

prints approximate values of the exponential function of the first ten integers.

#### FILES

```
/usr/lib/lib.b  mathematical library
/usr/bin/dc     desk calculator proper
```

#### SEE ALSO

*dc(1)*.

*BC—An Arbitrary Precision Desk-Calculator Language* by L. L. Cherry and R. Morris.

#### BUGS

No *&&*, *||* yet.

*For* statement must have all three E's.

*Quit* is interpreted when read, not when executed.

**NAME****bdiff** — big diff**SYNOPSIS****bdiff** file1 file2 [n] [-s]**DESCRIPTION**

*Bdiff* is used in a manner analogous to *diff*(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*. *Bdiff* ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. The value of *n* is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail. If *file1* (*file2*) is *-*, the standard input is read. The optional *-s* (silent) argument specifies that no diagnostics are to be printed by *bdiff* (note, however, that this does not suppress possible exclamations by *diff*). If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

**FILES**

/tmp/bd????

**SEE ALSO***diff*(1).**DIAGNOSTICS**Use *help*(1) for explanations.

## NAME

bfs — big file scanner

## SYNOPSIS

bfs [ - ] name

## DESCRIPTION

*Bfs* is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 255 characters per line. *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the *w* command. The optional *-* suppresses printing of sizes. Input is prompted with *\** if *P* and a carriage return are typed as in *ed*. Prompting can be turned off again by inputting another *P* and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols besides */* and *?*: *>* indicates downward search without wrap-around, and *<* indicates upward search without wrap-around. Since *bfs* uses a different regular expression-matching routine from *ed*, the regular expressions accepted are slightly wider in scope (see *regcmp*(3X)). There is a slight difference in mark names: only the letters *a* through *z* may be used, and all 26 marks are remembered.

The *e*, *g*, *v*, *k*, *n*, *p*, *q*, *w*, *=*, *!* and null commands operate as described under *ed*. Commands such as *---*, *+++*, *+++*, *=*, *-12*, and *+4p* are accepted. Note that *1,10p* and *1,10* will both print the first ten lines. The *f* command only prints the name of the file being scanned; there is no *remembered* file name. The *w* command is independent of output diversion, truncation, or crunching (see the *x0*, *xt* and *xc* commands, below). The following additional commands are available:

*xf file*

Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the *xf*. *Xf* commands may be nested to a depth of 10.

*xo [file]*

Further output from the *p* and null commands is diverted to the named *file*, which, if necessary, is created mode 666. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

*: label*

This positions a *label* in a command file. The *label* is terminated by new-line, and blanks between the *:* and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

*(. . .)xb/regular expression/label*

A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between 1 and \$.
2. The second address is less than the first.
3. The regular expression doesn't match at least one line in the specified range, including the first and last lines.

On success, `.` is set to the line matched and a jump is made to *label*. This command is the only one that doesn't issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

```
xb/^/ label
```

is an unconditional jump.

The `xb` command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

#### `xt number`

Output from the `p` and null commands is truncated to at most *number* characters. The initial number is 255.

#### `xv[digit][spaces][value]`

The variable name is the specified *digit* following the `xv`. `xv5100` or `xv5 100` both assign the value 100 to the variable 5. `Xv61,100p` assigns the value 1,100p to the variable 6. To reference a variable, put a `%` in front of the variable name. For example, using the above assignments for variables 5 and 6:

```
1,%5p
1,%5
%6
```

will all print the first 100 lines.

```
g/%5/p
```

would globally search for the characters 100 and print each line containing a match. To escape the special meaning of `%`, a `\` must precede it.

```
g/".*%[cds]/p
```

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.

Another feature of the `xv` command is that the first line of output from a UNIX command can be stored into a variable. The only requirement is that the first character of *value* be an `!`. For example:

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

would put the current line into variable 5, print it, and increment the variable 6 by one. To escape the special meaning of `!` as the first character of *value*, precede it with a `\`.

```
xv7\!date
```

stores the value `!date` into variable 7.

**xbz** *label*

**xbn** *label*

These two commands will test the last saved *return code* from the execution of a UNIX command (`!command`) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string size.

```
xv55
:l
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn l
xv45
:l
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz l
```

**xc** [*switch*]

If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0** it isn't. Without an argument, **xc** reverses *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

#### SEE ALSO

`csplit(1)`, `ed(1)`, `regcmp(3X)`.

#### DIAGNOSTICS

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

**NAME**

**bs** — a compiler/interpreter for modest-sized programs

**SYNOPSIS**

**bs** [ file [ args ] ]

**DESCRIPTION**

*Bs* is a remote descendant of Basic and Snobol4 with a little C language thrown in. *Bs* is designed for programming tasks where program development time is as important as the resulting speed of execution. Formalities of data declaration and file/process manipulation are minimized. Line-at-a-time debugging, the *trace* and *dump* statements, and useful run-time error messages all simplify program testing. Furthermore, incomplete programs can be debugged; *inner* functions can be tested before *outer* functions have been written and vice versa.

If the command line *file* argument is provided, the file is used for input before the console is read. By default, statements read from the file argument are compiled for later execution. Likewise, statements entered from the console are normally executed immediately (see *compile* and *execute* below). Unless the final operation is assignment, the result of an immediate expression statement is printed.

*Bs* programs are made up of input lines. If the last character on a line is a \, the line is continued. *Bs* accepts lines of the following form:

```
statement
label statement
```

A label is a *name* (see below) followed by a colon. A label and a variable can have the same name.

A *bs* statement is either an expression or a keyword followed by zero or more expressions. Some keywords (*clear*, *compile*, *!*, *execute*, *include*, *ibase*, *obase*, and *run*) are always executed as they are compiled.

**Statement Syntax:****expression**

The expression is executed for its side effects (value, assignment or function call). The details of expressions follow the description of statement types below.

**break**

*Break* exits from the inner-most *for/while* loop.

**clear**

Clears the symbol table and compiled statements. *Clear* is executed immediately.

**compile [ expression ]**

Succeeding statements are compiled (overrides the immediate execution default). The optional expression is evaluated and used as a file name for further input. A *clear* is associated with this latter case. *Compile* is executed immediately.

**continue**

*Continue* transfers to the loop-continuation of the current *for/while* loop.

**dump [ name ]**

The name and current value of every non-local variable is printed. Optionally, only the named variable is reported. After an error or interrupt, the number of the last statement and (possibly) the user-function trace are displayed.

**exit** [ expression ]

Return to system level. The expression is returned as process status.

**execute**

Change to immediate execution mode (an interrupt has a similar effect). This statement does not cause stored statements to execute (see *run* below).

**for** name = expression expression statement

**for** name = expression expression

...

**next**

**for** expression , expression , expression statement

**for** expression , expression , expression

...

**next**

The *for* statement repetitively executes a statement (first form) or a group of statements (second form) under control of a named variable. The variable takes on the value of the first expression, then is incremented by one on each loop, not to exceed the value of the second expression. The third and fourth forms require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action (normally an increment).

**fun** f( [ a, ... ] ) [ v, ... ]

...

**nuf**

*Fun* defines the function name, arguments, and local variables for a user-written function. Up to ten arguments and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested.

**freturn**

A way to signal the failure of a user-written function. See the interrogation operator (?) below. If interrogation is not present, *freturn* merely returns zero. When interrogation is active, *freturn* transfers to that expression (possibly by-passing intermediate function returns).

**goto** name

Control is passed to the internally stored statement with the matching label.

**ibase** *N*

*Ibase* sets the input base (radix) to *N*. The only supported values for *N* are 8, 10 (the default), and 16. Hexadecimal values 10–15 are entered as a–f. A leading digit is required (i.e., f0a must be entered as 0f0a). *Ibase* (and *obase*, below) are executed immediately.

**if** expression statement

**if** expression

...

[ **else**

... ]

**fi**

The statement (first form) or group of statements (second form) is executed if the expression evaluates to non-zero. The strings 0 and "" (null) evaluate as zero. In the second form, an optional *else* allows for a group of statements to be executed when the first group is not. The only statement permitted on the same line with an *else* is an *if*; only

other *fi*'s can be on the same line with a *fi*. The elision of *else* and *if* into an *elif* is supported. Only a single *fi* is required to close an *if ... elif ... [ else ... ]* sequence.

**include expression**

The expression must evaluate to a file name. The file must contain *bs* source statements. Such statements become part of the program being compiled. *Include* statements may not be nested.

**obase *N***

*Obase* sets the output base to *N* (see *ibase* above).

**onintr label**

**onintr**

The *onintr* command provides program control of interrupts. In the first form, control will pass to the label given, just as if a *goto* had been executed at the time *onintr* was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt will cause *bs* to terminate.

**return [expression]**

The expression is evaluated and the result is passed back as the value of a function call. If no expression is given, zero is returned.

**run**

The random number generator is reset. Control is passed to the first internal statement. If the *run* statement is contained in a file, it should be the last statement.

**stop**

Execution of internal statements is stopped. *Bs* reverts to immediate mode.

**trace [ expression ]**

The *trace* statement controls function tracing. If the expression is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns will be printed. Each *return* decrements the *trace* expression value.

**while expression statement**

**while expression**

...

**next**

*While* is similar to *for* except that only the conditional expression for loop-continuation is given.

**! shell command**

An immediate escape to the Shell.

**# ...**

This statement is ignored. It is used to interject commentary in a program.

**Expression Syntax:**

**name**

A name is used to specify a variable. Names are composed of a letter (upper or lower case) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared in *fun* statements, all names are global to the program. Names can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function *open()* below).



**name** ( [expression [ , expression] ... ] )

Functions can be called by a name followed by the arguments in parentheses separated by commas. Except for built-in functions (listed below), the name must be defined with a *fun* statement. Arguments to functions are passed by value.

**name** [ expression [ , expression ] ... ]

This syntax is used to reference either arrays or tables (see built-in *table* functions below). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference is syntactically identical to a name; **a[1,2]** is the same as **a[1][2]**. The truncated expressions are restricted to values between 0 and 32767.

**number**

A number is used to represent a constant value. A number is written in Fortran style, and contains digits, an optional decimal point, and possibly a scale factor consisting of an *e* followed by a possibly signed exponent.

**string**

Character strings are delimited by `"` characters. The `\` escape character allows the double quote (`\`), new-line (`\n`), carriage return (`\r`), back-space (`\b`), and tab (`\t`) characters to appear in a string. Otherwise, `\` stands for itself.

( expression )

Parentheses are used to alter the normal order of evaluation.

( expression, expression [ , expression ... ] ) [ expression ]

The bracketed expression is used as a subscript to select a comma-separated expression from the parenthesized list. List elements are numbered from the left, starting at zero. The expression:

( False, True ) [ a == b ]

has the value **True** if the comparison is true.

? expression

The interrogation operator tests for the success of the expression rather than its value. At the moment, it is useful for testing end-of-file (see examples in the *Programming Tips* section below), the result of the *eval* built-in function, and for checking the return from user-written functions (see *freturn*). An interrogation "trap" (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

- expression

The result is the negation of the expression.

++ name

Increments the value of the variable (or array reference). The result is the new value.

-- name

Decrements the value of the variable. The result is the new value.

! expression

The logical negation of the expression. Watch out for the shell escape command.

expression *operator* expression

Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands are

converted to numeric form before the function is applied.

### Binary Operators (in increasing precedence):

=

= is the assignment operator. The left operand must be a name or an array element. The result is the right operand. Assignment binds right to left, all other operators bind left to right.

-

\_ (underscore) is the concatenation operator.

& |

& (logical and) has result zero if either of its arguments are zero. It has result one if both of its arguments are non-zero; | (logical or) has result zero if both of its arguments are zero. It has result one if either of its arguments is non-zero. Both operators treat a null string as a zero.

< <= > >= == !=

The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, != not equal to) return one if their arguments are in the specified relation. They return zero otherwise. Relational operators at the same level extend as follows:  $a > b > c$  is the same as  $a > b$  &  $b > c$ . A string comparison is made if both operands are strings.

+ -

Add and subtract.

\* / %

Multiply, divide, and remainder.

^

Exponentiation.

### Built-in Functions:

#### *Dealing with arguments*

**arg(i)**

is the value of the  $i$ -th actual parameter on the current level of function call. At level zero, *arg* returns the  $i$ -th command-line argument (*arg*(0) returns *bs*).

**narg()**

returns the number of arguments passed. At level zero, the command argument count is returned.

#### *Mathematical*

**abs(x)**

is the absolute value of  $x$ .

**atan(x)**

is the arctangent of  $x$ . Its value is between  $-\pi/2$  and  $\pi/2$ .

**ceil(x)**

returns the smallest integer not less than  $x$ .

**cos(x)**

is the cosine of  $x$  (radians).

**exp(x)**

is the exponential function of  $x$ .

**floor(x)**

returns the largest integer not greater than  $x$ .

**log(x)**  
is the natural logarithm of *x*.

**rand()**  
is a uniformly distributed random number between zero and one.

**sin(x)**  
is the sine of *x* (radians).

**sqrt(x)**  
is the square root of *x*.

### String operations

**size(s)**  
the size (length in bytes) of *s* is returned.

**format(f, a)**  
returns the formatted value of *a*. *F* is assumed to be a format specification in the style of *printf*(3S). Only the *%...f*, *%...e*, and *%...s* types are safe.

**index(x, y)**  
returns the number of the first position in *x* that any of the characters from *y* matches. No match yields zero.

**trans(s, f, t)**  
Translates characters of the source *s* from matching characters in *f* to a character in the same position in *t*. Source characters that do not appear in *f* are copied to the result. If the string *f* is longer than *t*, source characters that match in the excess portion of *f* do not appear in the result.

**substr(s, start, width)**  
returns the sub-string of *s* defined by the *starting* position and *width*.

**match(string, pattern)**

**mstring(n)**  
The *pattern* is similar to the regular expression syntax of the *ed*(1) command. The characters *.*, *[*, *]*, *^* (inside brackets), *\** and *\$* are special. The *mstring* function returns the *n*-th ( $1 \leq n \leq 10$ ) substring of the subject that occurred between pairs of the pattern symbols *\(* and *\)* for the most recent call to *match*. To succeed, patterns must match the beginning of the string (as if all patterns began with *^*). The function returns the number of characters matched. For example:

```
match("a123ab123", ".*\([a-z]\)") == 6
mstring(1) == "b"
```

### File handling

**open(name, file, function)**

**close(name)**

The *name* argument must be a *bs* variable name (passed as a string). For the *open*, the *file* argument may be 1) a 0 (zero), 1, or 2 representing standard input, output, or error output, respectively, 2) a string representing a file name, or 3) a string beginning with an *!* representing a command to be executed (via *sh -c*). The *function* argument must be either *r* (read), *w* (write), *W* (write without new-line), or *a* (append). After a *close*, the *name* reverts to being an ordinary variable. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

Examples are given in the following section.

**access(s, m)**  
executes *access*(2).

**ftype(s)**  
returns a single character file type indication: **f** for regular file, **p** for FIFO (i.e., named pipe), **d** for directory, **b** for block special, or **c** for character special.

### Tables

**table(name, size)**

A table in *bs* is an associatively accessed, single-dimension array. "Subscripts" (called keys) are strings (numbers are converted). The *name* argument must be a *bs* variable name (passed as a string). The *size* argument sets the minimum number of elements to be allocated. *Bs* prints an error message and stops on table overflow.

**item(name, i)**

**key()**

The *item* function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the *item* function accesses values, the *key* function accesses the "subscript" of the previous *item* call. The *name* argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table, for example:

```
table("t", 100)
...
# If word contains "party", the following expression adds one
# to the count of that word:
++t[word]
...
# To print out the the key/value pairs:
for i = 0, (?s = item(t, i)), ++i if key() put = key()_"_s
```

**iskey(name, word)**

The *iskey* function tests whether the key word exists in the table *name* and returns one for true, zero for false.

### Odds and ends

**eval(s)**

The string argument is evaluated as a *bs* expression. The function is handy for converting numeric strings to numeric internal form. *Eval* can also be used as a crude form of indirection, as in:

```
name = 'xyz'
eval("++" _ name)
```

which increments the variable *xyz*. In addition, *eval* preceded by the interrogation operator permits the user to control *bs* error conditions. For example:

```
?eval("open(\"X\", \"XXX\", \"r\")")
```

returns the value zero if there is no file named "XXX" (instead of halting the user's program). The following executes a *goto* to the label *L* (if it exists):

```
label="L"
if !(?eval("goto " _ label)) puterr = "no label"
```

**plot(request, args)**

The *plot* function produces output on devices recognized by *tplot(1G)*. The *requests* are as follows:

| <i>Call</i>                     | <i>Function</i>                                                                                                                                                                 |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| plot(0, term)                   | causes further <i>plot</i> output to be piped into <i>tplot(1G)</i> with an argument of <i>-Term</i> .                                                                          |
| plot(4)                         | “erases” the plotter.                                                                                                                                                           |
| plot(2, string)                 | labels the current point with <i>string</i> .                                                                                                                                   |
| plot(3, x1, y1, x2, y2)         | draws the line between (x1,y1) and (x2,y2).                                                                                                                                     |
| plot(4, x, y, r)                | draws a circle with center (x,y) and radius <i>r</i> .                                                                                                                          |
| plot(5, x1, y1, x2, y2, x3, y3) | draws an arc (counterclockwise) with center (x1,y1) and endpoints (x2,y2) and (x3,y3).                                                                                          |
| plot(6)                         | is not implemented.                                                                                                                                                             |
| plot(7, x, y)                   | makes the current point (x,y).                                                                                                                                                  |
| plot(8, x, y)                   | draws a line from the current point to (x,y).                                                                                                                                   |
| plot(9, x, y)                   | draws a point at (x,y).                                                                                                                                                         |
| plot(10, string)                | sets the line mode to <i>string</i> .                                                                                                                                           |
| plot(11, x1, y1, x2, y2)        | makes (x1,y1) the lower left corner of the plotting area and (x2,y2) the upper right corner of the plotting area.                                                               |
| plot(12, x1, y1, x2, y2)        | causes subsequent x (y) coordinates to be multiplied by x1 (y1) and then added to x2 (y2) before they are plotted. The initial scaling is <b>plot(12, 1.0, 1.0, 0.0, 0.0)</b> . |

Some requests do not apply to all plotters. All requests except zero and twelve are implemented by piping characters to *tplot(1G)*. See *plot(4)* for more details.

**last()**

in immediate mode, *last* returns the most recently computed value.

**PROGRAMMING TIPS**

Using *bs* as a calculator:

```
$ bs
# Distance (inches) light travels in a nanosecond.
186000 * 5280 * 12 / 1e9
11.78496
...

# Compound interest (6% for 5 years on $1,000).
int = .06 / 4
bal = 1000
for i = 1 5*4 bal = bal + bal*int
bal - 1000
```

346.855007

```
...
exit
```

The outline of a typical *bs* program:

```
# initialize things:
var1 = 1
open("read", "infile", "r")
...
# compute:
while ?(str = read)
    ...
next
# clean up:
close("read")
...
# last statement executed (exit or stop):
exit
# last input line:
run
```

Input/Output examples:

```
# Copy "oldfile" to "newfile".
open("read", "oldfile", "r")
open("write", "newfile", "w")
...
while ?(write = read)
...
# close "read" and "write":
close("read")
close("write")

# Pipe between commands.
open("ls", "!ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls) ...
...
# be sure to close (wait for) these:
close("ls")
close("pr")
```

#### SEE ALSO

ed(1), sh(1), tplot(1G), access(2), printf(3S), stdio(3S), plot(4).  
See Section 3 of this volume for further description of the mathematical functions (*pow* on *exp*(3M) is used for exponentiation); *bs* uses the Standard Input/Output package.

**NAME**

cal — print calendar

**SYNOPSIS**

cal [ month ] year

**DESCRIPTION**

*Cal* prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

**BUGS**

The year is always considered to start in January even though this is historically naive.

Beware that “cal 78” refers to the early Christian era, not the 20th century.

**NAME**

calendar — reminder service

**SYNOPSIS**

calendar [ - ]

**DESCRIPTION**

*Calendar* consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Dec. 7," "december 7," "12/7," etc., are recognized, but not "7 December" or "7/12". On weekends "tomorrow" extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file **calendar** in their login directory and sends them any positive results by *mail*(1). Normally this is done daily by facilities in the UNIX operating system.

**FILES**

calendar  
/usr/lib/calprog to figure out today's and tomorrow's dates  
/etc/passwd  
/tmp/cal\*

**SEE ALSO**

mail(1).

**BUGS**

Your calendar must be public information for you to get reminder service. *Calendar's* extended idea of "tomorrow" does not account for holidays.



**NAME**

`cat` — concatenate and print files

**SYNOPSIS**

`cat [ -u ] [ -s ] file ...`

**DESCRIPTION**

*Cat* reads each *file* in sequence and writes it on the standard output. Thus:

```
cat file
```

prints the file, and:

```
cat file1 file2 >file3
```

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

If no input file is given, or if the argument `-` is encountered, *cat* reads from the standard input file. Output is buffered unless the `-u` option is specified. The `-s` option makes *cat* silent about non-existent files. No input file may be the same as the output file unless it is a special file.

**WARNING**

Command formats such as

```
cat file1 file2 >file1
```

will cause the original data in *file1* to be lost, therefore, take care when using shell special characters.

**SEE ALSO**

`cp(1)`, `pr(1)`.

**NAME**

cb - C program beautifier

**SYNOPSIS**

cb [ -s ] [ -j ] [ -l leng ] [ file ... ]

**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. Under default options, *cb* preserves all user new-lines. Under the *-s* flag *cb* canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*. The *-j* flag causes split lines to be put back together. The *-l* flag causes *cb* to split lines that are longer than *leng*.

**SEE ALSO**

cc(1).

*The C Programming Language* by B. W. Kernighan and D. M. Ritchie.

**BUGS**

Punctuation that is hidden in preprocessor statements will cause indentation errors.

## NAME

`cc`, `pcc` — C compiler

## SYNOPSIS

`cc` [ option ] ... file ...  
`pcc` [ option ] ... file ...

## DESCRIPTION

`Cc` is the UNIX C compiler. `Pcc` is the portable version for a PDP-11 machine. They accept several types of arguments:

Arguments whose names end with `.c` are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with `.o` substituted for `.c`. The `.o` file is normally deleted, however, if a single C program is compiled and loaded all at one go.

In the same way, arguments whose names end with `.s` are taken to be assembly source programs and are assembled, producing a `.o` file.

The following options are interpreted by `cc` and `pcc`. See `ld(1)` for link editor options and `cpp(1)` for more preprocessor options.

- `c` Suppress the link edit phase of the compilation, and force an object file to be produced even if only one program is compiled.
- `p` Arrange for the compiler to produce code which counts the number of times each routine is called; also, if link editing takes place, replace the standard startoff routine by one which automatically calls `monitor(3C)` at the start and arranges to write out a `mon.out` file at normal termination of execution of the object program. An execution profile can then be generated by use of `prof(1)`.
- `f` Link the object program with the floating-point interpreter for systems without hardware floating-point.
- `g` Cause the compiler to generate additional information needed for the use of `sdb(1)`. (Not for PDP-11.)
- `O` Invoke an object-code optimizer.
- `S` Compile the named C programs, and leave the assembler-language output on corresponding files suffixed `.s`.
- `E` Run only `cpp(1)` on the named C programs, and send the result to the standard output.
- `P` Run only `cpp(1)` on the named C programs, and leave the result on corresponding files suffixed `.i`.
- `Bstring`  
Construct pathnames for substitute compiler, assembler and link editor passes by concatenating `string` with the suffixes `cpp`, `c0` (or `ccom` or `comp`, see under FILES below), `c1`, `c2`, `as` and `ld`. If `string` is empty it is taken to be `/lib/o`.
- `t[p012a]`  
Find only the designated compiler, assembler and link editor passes in the files whose names are constructed by a `—B` option. In the absence of a `—B` option, the `string` is taken to be `/lib/n`.  
—`t ""` is equivalent to `—tp012`.
- `Wc,arg1[,arg2...]`  
Hand off the argument[s] `argi` to pass `c` where `c` is one of `[p012a]` indicating preprocessor, compiler first pass, compiler second pass,

optimizer, assembler, or link editor, respectively.

- d This option is no longer allowed because of a conflict of meaning. The -W option must be used to specify precisely its destination. To indicate the -dn option for the VAX assembler, use -Wa,-dn. To indicate the -d option for the link editor, use -Wl,-d.

Other arguments are taken to be either link editor option arguments, C preprocessor option arguments, or C-compatible object programs, typically produced by an earlier *cc* or *pcc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the name **a.out**.

## FILES

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| file.c        | input file                                                             |
| file.o        | object file                                                            |
| a.out         | linked output                                                          |
| /tmp/ctm*     | temporary                                                              |
| /lib/cpp      | C preprocessor <i>cpp</i> (1)                                          |
| /lib/c[01]    | PDP-11 compiler, <i>cc</i>                                             |
| /usr/lib/comp | compiler, <i>pcc</i>                                                   |
| /lib/ccom     | VAX compiler, <i>cc</i>                                                |
| /lib/c2       | optional optimizer                                                     |
| /lib/oc*      | backup compiler, <i>occ</i>                                            |
| /lib/nc*      | test compiler, <i>ncc</i>                                              |
| /bin/as       | assembler, <i>as</i> (1)                                               |
| /bin/ld       | link editor, <i>ld</i> (1)                                             |
| /lib/crt0.o   | runtime startoff                                                       |
| /lib/mcrt0.o  | startoff for profiling                                                 |
| /lib/fcrt0.o  | startoff for floating-point interpretation (PDP-11 only)               |
| /lib/fmcrt0.o | startoff for floating-point interpretation and profiling (PDP-11 only) |
| /lib/libc.a   | standard library, see (3)                                              |

## SEE ALSO

*The C Programming Language* by B. W. Kernighan and D. M. Ritchie.  
*Programming in C—A Tutorial* by B. W. Kernighan.  
*C Reference Manual* by D. M. Ritchie.  
 adb(1), cpp(1), as(1), ld(1), prof(1), sdb(1), monitor(3C).

## DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or the link editor. Of these, the most mystifying are from the PDP-11 assembler, in particular **m**, which means a multiply-defined external symbol (function or data).

**NAME**

`cd` — change working directory

**SYNOPSIS**

`cd` [ *directory* ]

**DESCRIPTION**

If *directory* is not specified, the value of shell parameter `$HOME` is used as the new working directory. If *directory* specifies a complete path starting with `/`, `.`, `..`, *directory* becomes the new working directory. If neither case applies, `cd` tries to find the designated directory relative to one of the paths specified by the `$CDPATH` shell variable. `$CDPATH` has the same syntax as, and similar semantics to, the `$PATH` shell variable. `Cd` must have execute (search) permission in *directory*.

Because a new process is created to execute each command, `cd` would be ineffective if it were written as a normal command; therefore, it is recognized and internal to the shell.

**SEE ALSO**

`pwd(1)`, `sh(1)`, `chdir(2)`.

## NAME

`cdc` — change the delta commentary of an SCCS delta

## SYNOPSIS

`cdc -rSID [-m[mrlist]] [-y[comment]] files`

## DESCRIPTION

`Cdc` changes the *delta commentary*, for the *SID* specified by the `-r` keyletter, of each named SCCS file.

*Delta commentary* is defined to be the Modification Request (MR) and comment information normally specified via the *delta*(1) command (`-m` and `-y` keyletters).

If a directory is named, `cdc` behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with `s.`) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to `cdc`, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

`-rSID` Used to specify the SCCS *IDentification (SID)* string of a delta for which the delta commentary is to be changed.

`-m[mrlist]` If the SCCS file has the `v` flag set (see *admin*(1)) then a list of MR numbers to be added and/or deleted in the delta commentary of the *SID* specified by the `-r` keyletter *may* be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of *delta*(1). In order to delete an MR, precede the MR number with the character `!` (see *EXAMPLES*). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a "comment" line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If `-m` is not used and the standard input is a terminal, the prompt `MRs?` is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The `MRs?` prompt always precedes the `comments?` prompt (see `-y` keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the `v` flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, `cdc` terminates

and the delta commentary remains unchanged.

**-y[comment]** Arbitrary text used to replace the *comment*(s) already existing for the delta specified by the **-r** keyletter. The previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

The exact permissions necessary to modify the SCCS file are documented in the *Source Code Control System User's Guide*. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

#### EXAMPLES

```
cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrouble s.file
```

adds bl78-12345 and bl79-00001 to the MR list, removes bl77-54321 from the MR list, and adds the comment **trouble** to delta 1.6 of s.file.

```
cdc -r1.6 s.file
MRs? !bl77-54321 bl78-12345 bl79-00001
comments? trouble
```

does the same thing.

#### WARNINGS

If SCCS file names are supplied to the *cdc* command via the standard input (**-** on the command line), then the **-m** and **-y** keyletters must also be used.

#### FILES

x-file (see *delta*(1))  
z-file (see *delta*(1))

#### SEE ALSO

*admin*(1), *delta*(1), *get*(1), *help*(1), *prs*(1), *scsfile*(4).  
*Source Code Control System User's Guide* in the *UNIX System User's Guide*.

#### DIAGNOSTICS

Use *help*(1) for explanations.

## NAME

`cflow` — generate C flow graph

## SYNOPSIS

`cflow` [`-r`] [`-ix`] [`-i_`] [`-dnum`] files

## 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`, `.l`, `.c`, and `.i` are YACC'd, LEX'd, and C-preprocessed (bypassed for `.i` files) as appropriate and then run through the first pass of *lint*(1). (The `-I`, `-D`, and `-U` options of the C-preprocessor are also understood.) Files suffixed with `.s` are assembled and information is extracted (as in `.o` files) from the symbol table. 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 (i.e., 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 (e.g., `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 (e.g., *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.

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

```
int    i;

main()
{
    f();
    g();
    f();
}

f()
{
    i = h();
}
```

the command

```
cflow file.c
```

produces the 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(1)` can be used to compress the tab expansion to something less than every eight spaces.

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 flow graph.
- `-i_` Include names that begin with an underscore. The default is to exclude these functions (and data if `-ix` is used).
- `-dnum` The *num* decimal integer indicates the depth at which the flow graph 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.

#### DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C-preprocessor).

#### SEE ALSO

`as(1)`, `cc(1)`, `lex(1)`, `lint(1)`, `nm(1)`, `pr(1)`, `yacc(1)`.

#### BUGS

Files produced by `lex(1)` and `yacc(1)` cause the reordering of line number declarations which can confuse `cflow`. To get proper results, feed `cflow` the `yacc` or `lex` input.

**NAME**

chmod — change mode

**SYNOPSIS**

chmod mode files

**DESCRIPTION**

The permissions of the named *files* are changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

|      |                                         |
|------|-----------------------------------------|
| 4000 | set user ID on execution                |
| 2000 | set group ID on execution               |
| 1000 | sticky bit, see <i>chmod(2)</i>         |
| 0400 | read by owner                           |
| 0200 | write by owner                          |
| 0100 | execute (search in directory) by owner  |
| 0070 | read, write, execute (search) by group  |
| 0007 | read, write, execute (search) by others |

A symbolic *mode* has the form:

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

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, the default if *who* is omitted.

*Op* can be **+** to add *permission* to the file's mode, **-** to take away *permission*, or **=** to assign *permission* absolutely (all other bits will be reset).

*Permission* is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group ID) and **t** (save text, or sticky); **u**, **g**, or **o** indicate that *permission* is to be taken from the current mode. 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. The letter **s** is only useful with **u** or **g** and **t** only works with **u**.

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

**EXAMPLES**

The first example denies write permission to others, the second makes a file executable:

```
chmod o-w file
```

```
chmod +x file
```

**SEE ALSO**

ls(1), chmod(2).



**NAME**

chown, chgrp — change owner or group

**SYNOPSIS**

**chown** owner file ...

**chgrp** group file ...

**DESCRIPTION**

*Chown* changes the owner of the *files* to *owner*. The owner may be either a decimal user ID or a login name found in the password file.

*Chgrp* changes the group ID of the *files* to *group*. The group may be either a decimal group ID or a group name found in the group file.

**FILES**

/etc/passwd

/etc/group

**SEE ALSO**

chown(2), group(4), passwd(4).

**NAME**

`cmp` — compare two files

**SYNOPSIS**

`cmp` [ `-l` ] [ `-s` ] `file1 file2`

**DESCRIPTION**

The two files are compared. (If `file1` is `-`, the standard input is used.) Under default options, `cmp` makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

**Options:**

- `-l` Print the byte number (decimal) and the differing bytes (octal) for each difference.
- `-s` Print nothing for differing files; return 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.

**NAME**

*col* — filter reverse line-feeds

**SYNOPSIS**

*col* [ **-b***fp*x ]

**DESCRIPTION**

*Col* reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code ESC-7), and by forward and reverse half-line-feeds (ESC-9 and ESC-8). *Col* is particularly useful for filtering multicolumn output made with the *.rt* command of *nroff* and output resulting from use of the *tbl*(1) preprocessor.

If the **-b** option is given, *col* assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read will be output.

Although *col* accepts half-line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. This treatment can be suppressed by the **-f** (fine) option; in this case, the output from *col* may contain forward half-line-feeds (ESC-9), but will still never contain either kind of reverse line motion.

Unless the **-x** option is given, *col* will convert white space to tabs on output wherever possible to shorten printing time.

The ASCII control characters SO (\017) and SI (\016) are assumed by *col* to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output SI and SO characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, new-line, SI, SO, VT (\013), and ESC followed by 7, 8, or 9. The VT character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

Normally, *col* will ignore any unknown to it escape sequences found in its input; the **-p** option may be used to cause *col* to output these sequences as regular characters, subject to overprinting from reverse line motions. The use of this option is highly discouraged unless the user is fully aware of the textual position of the escape sequences.

**SEE ALSO**

*nroff*(1), *tbl*(1).

**NOTES**

The input format accepted by *col* matches the output produced by *nroff* with either the **-T37** or **-Tlp** options. Use **-T37** (and the **-f** option of *col*) if the ultimate disposition of the output of *col* will be a device that can interpret half-line motions, and **-Tlp** otherwise.

**BUGS**

Cannot back up more than 128 lines.

Allows at most 800 characters, including backspaces, 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

comb — combine SCCS deltas

## SYNOPSIS

comb [-o] [-s] [-psid] [-clist] files

## DESCRIPTION

*Comb* generates a shell procedure (see *sh*(1)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The generated shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

- psid The SCCS IDentification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.
- clist A list (see *get*(1) for the syntax of a list) of deltas to be preserved. All other deltas are discarded.
- o For each *get -e* generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the -o keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.
- s This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

$$100 * (\text{original} - \text{combined}) / \text{original}$$

It is recommended that before any SCCS files are actually combined, one should use this option to determine exactly how much space is saved by the combining process.

If no keyletter arguments are specified, *comb* will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

## FILES

s.COMB           The name of the reconstructed SCCS file.  
comb?????       Temporary.

## SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).  
*Source Code Control System User's Guide* in the *UNIX System User's Guide*.

## DIAGNOSTICS

Use *help*(1) for explanations.

## 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**

comm — select or reject lines common to two sorted files

**SYNOPSIS**

**comm** [ - [ 123 ] ] file1 file2

**DESCRIPTION**

*Comm* reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort(1)*), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name — means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus **comm -12** prints only the lines common to the two files; **comm -23** prints only lines in the first file but not in the second; **comm -123** is a no-op.

**SEE ALSO**

cmp(1), diff(1), sort(1), uniq(1).

**NAME**

convert — convert object and archive files to common formats

**SYNOPSIS**

convert infile outfile

**DESCRIPTION**

*Convert* transforms input *infile* to output *outfile*. *Infile* must be different from *outfile*. *Infile* may be any one of the following:

- 1) a pre-UNIX 5.0 VAX object file or link edited (a.out) module
- 2) a pre-UNIX 5.0 VAX archive of object files or link edited (a.out) modules
- 3) a pre-UNIX 5.0 3B20S archive of object files or link edited (a.out) modules.

*Convert* will transform *infile* to one of the following:

- 1) an equivalent UNIX 5.0 VAX object file or link edited (a.out) module
- 2) an equivalent UNIX 5.0 portable archive of equivalent object files or link edited (a.out) modules
- 3) an equivalent UNIX 5.0 portable archive of unaltered 3B20S object files or link edited (a.out) modules.

All other types of input to the *convert*(1) command will be passed unmodified from the input file to the output file (along with appropriate warning messages). When transforming archive files, the *convert*(1) command will inform the user that the archive symbol table has been deleted. The archive symbol table may be restored by executing the *ar*(1) command with the *s* option.

The *convert* command may be used in conjunction with the *arcv*(1) command to transform archives generated on a PDP-11 to the UNIX 5.0 archive format for usage on a 3B20S or VAX processor.

**FILES**

/tmp/conv\*

**SEE ALSO**

ar(1), arcv(1), a.out(4), ar(4).





**NAME**

**cp**, **ln**, **mv** — copy, link or move files

**SYNOPSIS**

```
cp file1 [ file2 ...] target
ln file1 [ file2 ...] target
mv file1 [ file2 ...] target
```

**DESCRIPTION**

*File1* is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same (take care when using *sh*(1) metacharacters). If *target* is a directory, then one or more files are copied (linked, moved) to that directory.

If *mv* determines that the mode of *target* forbids writing, it will print the mode (see *chmod*(2)) and read the standard input for one line (if the standard input is a terminal); if the line begins with *y*, the move takes place; if not, *mv* exits.

Only *mv* will allow *file1* to be a directory, in which case the directory rename will occur only if the two directories have the same parent.

**SEE ALSO**

*cpio*(1), *rm*(1), *chmod*(2).

**BUGS**

If *file1* and *target* lie on different file systems, *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.

*Ln* will not link across file systems.

## NAME

**cpio** — copy file archives in and out

## SYNOPSIS

**cpio -o** [ **acBv** ]

**cpio -i** [ **BcdmrtuvfsSb6** ] [ **patterns** ]

**cpio -p** [ **adlmruv** ] **directory**

## DESCRIPTION

**Cpio -o** (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information.

**Cpio -i** (copy in) extracts files from the standard input which is assumed to be the product of a previous **cpio -o**. Only files with names that match *patterns* are selected. *Patterns* are given in the name-generating notation of *sh*(1). In *patterns*, meta-characters **?**, **\***, and **[...]** match the slash / character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is **\*** (i.e., select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below.

**Cpio -p** (*pass*) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are:

- a** Reset access times of input files after they have been copied.
- B** Input/output is to be blocked 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from **/dev/rmt?**).
- d** *Directories* are to be created as needed.
- c** Write *header* information in ASCII character form for portability.
- r** Interactively *rename* files. If the user types a null line, the file is skipped.
- t** Print a *table of contents* of the input. No files are created.
- u** Copy *unconditionally* (normally, an older file will not replace a newer file with the same name).
- v** *Verbose*: causes a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an **ls -l** command (see *ls*(1)).
- l** Whenever possible, link files rather than copying them. Usable only with the **-p** option.
- m** Retain previous file modification time. This option is ineffective on directories that are being copied.
- f** Copy in all files except those in *patterns*.
- s** Swap bytes. Use only with the **-i** option.
- S** Swap halfwords. Use only with the **-i** option.
- b** Swap both bytes and halfwords. Use only with the **-i** option.
- 6** Process an old (i.e., UNIX *Sixth* Edition format) file. Only useful with **-i** (copy in).

## EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/mt0
cd olddir
find . -depth -print | cpio -pdl newdir
```

The trivial case “find . -depth -print | cpio -oB >/dev/rmt0” can be handled more efficiently by:

```
find . -cpio /dev/rmt0
```

**SEE ALSO**

ar(1), find(1), cpio(4).

**BUGS**

Path names are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the super-user can copy special files. The **-B** option does not work with certain magnetic tape drives (see *un32(7)* in the *UNIX System Administrator's Manual*).

## NAME

cpp — the C language preprocessor

## SYNOPSIS

`/lib/cpp [ option ... ] [ ifile [ ofile ] ]`

## DESCRIPTION

*Cpp* is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc(1)* command. Thus the output of *cpp* is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, *cpp* and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of *cpp* other than in this framework is not suggested. The preferred way to invoke *cpp* is through the *cc(1)* command since the functionality of *cpp* may someday be moved elsewhere. See *m4(1)* for a general macro processor.

*Cpp* optionally accepts two file names as arguments. *Ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following *options* to *cpp* are recognized:

- P Preprocess the input without producing the line control information used by the next pass of the C compiler.
- C By default, *cpp* strips C-style comments. If the —C option is specified, all comments (except those found on *cpp* directive lines) are passed along.

—U*name*

Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

|                   |                                  |
|-------------------|----------------------------------|
| operating system: | ibm, gcos, os, tss, unix         |
| hardware:         | interdata, pdp11, u370, u3b, vax |
| UNIX variant:     | RES, RT                          |

—D*name*—D*name*=*def*

Define *name* as if by a **#define** directive. If no =*def* is given, *name* is defined as 1.

—I*dir*

Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in "" will be searched for first in the directory of the *ifile* argument, then in directories named in —I options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the *ifile* argument is not searched.

Two special names are understood by *cpp*. The name `__LINE__` is defined as the current line number (as a decimal integer) as known by *cpp*, and `__FILE__` is defined as the current file name (as a C string) as known by *cpp*. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directives start with lines begun by **#**. The directives are:

**#define** *name token-string*

Replace subsequent instances of *name* with *token-string*.

**#define** *name( arg, ..., arg ) token-string*

Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma

separated tokens, and a ) by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma separated list.

**#undef name**

Cause the definition of *name* (if any) to be forgotten from now on.

**#include "filename"**

**#include <filename>**

Include at this point the contents of *filename* (which will then be run through *cpp*). When the <*filename*> notation is used, *filename* is only searched for in the standard places. See the -I option above for more detail.

**#line integer-constant "filename"**

Causes *cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file where it comes from. If "*filename*" is not given, the current file name is unchanged.

**#endif**

Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

**#ifdef name**

The lines following will appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

**#ifndef name**

The lines following will not appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

**#if constant-expression**

Lines following will appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the ?: operator, the unary -, !, and ~ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined ( name )** or **defined name**. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

**#else** Reverses the notion of the test directive which matches this directive. So if lines previous to this directive are ignored, the following lines will appear in the output. And vice versa.

The test directives and the possible **#else** directives can be nested.

**FILES**

/usr/include                      standard directory for **#include** files

**SEE ALSO**

cc(1), m4(1).

**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, previous versions of *cpp* put out the newlines as they were found and expanded. The current version of *cpp* replaces these newlines with blanks to alleviate problems that the previous versions had when this occurred.

**NAME**

**cprs** — compress an IS25 object file

**SYNOPSIS**

**cprs** [-**pv**] *file1* *file2*

**DESCRIPTION**

The *cprs* command reduces the size of an IS25 object file, *file1*, by removing duplicate structure and union descriptors. The reduced file, *file2*, is produced as output.

The options are:

- p** Print statistical messages including: total number of tags, total duplicate tags, and total reduction of *file1*.
- v** Print verbose error messages if error condition occurs.

**SEE ALSO**

*strip*(1).

**NAME**

crypt — encode/decode

**SYNOPSIS**

crypt [ password ]

**DESCRIPTION**

*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:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

will print the clear.

Files encrypted by *crypt* are compatible with those treated by the editor *ed* 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 clear text 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 known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e. 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. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. The choice of keys and key security are the most vulnerable aspect of *crypt*.

**FILES**

/dev/tty            for typed key

**SEE ALSO**

ed(1), makekey(1).

**BUGS**

If output is piped to *nroff* and the encryption key is *not* given on the command line, *crypt* can leave terminal modes in a strange state (see *stty*(1)). If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.



## NAME

`csplit` - context split

## SYNOPSIS

`csplit` [-s] [-k] [-f prefix] file arg1 [... argn]

## DESCRIPTION

*Csplit* reads *file* and separates it into  $n+1$  sections, defined by the arguments *arg1* ... *argn*. By default the sections are placed in `xx00` ... `xxn` ( $n$  may not be greater than 99). These sections get the following pieces of *file*:

- 00: From the start of *file* up to (but not including) the line referenced by *arg1*.
- 01: From the line referenced by *arg1* up to the line referenced by *arg2*.
- ⋮
- $n+1$ : From the line referenced by *argn* to the end of *file*.

The options to *csplit* are:

- s *Csplit* normally prints the character counts for each file created. If the -s option is present, *csplit* suppresses the printing of all character counts.
- k *Csplit* normally removes created files if an error occurs. If the -k option is present, *csplit* leaves previously created files intact.
- f *prefix* If the -f option is used, the created files are named *prefix00* ... *prefixn*. The default is `xx00` ... `xxn`.

The arguments (*arg1* ... *argn*) to *csplit* can be a combination of the following:

- /regexp/* A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *regexp*. The current line becomes the line containing *regexp*. This argument may be followed by an optional + or - some number of lines (e.g., `/Page/-5`).
- %regexp%* This argument is the same as */regexp/*, except that no file is created for the section.
- lnno* A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.
- {num}* Repeat argument. This argument may follow any of the above arguments. If it follows a *regexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

Enclose all *regexp* type arguments that contain blanks or other characters meaningful to the Shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *Csplit* does not affect the original file; it is the users responsibility to remove it.

## EXAMPLES

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

This example creates four files, `cobol00` ... `cobol03`. After editing the "split" files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

```
csplit -k file 100 {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The `-k` option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%' '/')/+1' {20}
```

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`.

**SEE ALSO**

`ed(1)`, `sh(1)`, `regexp(5)`.

**DIAGNOSTICS**

Self explanatory except for:

`arg` — out of range

which means that the given argument did not reference a line between the current position and the end of the file.

**NAME**

*ct* - spawn *getty* to a remote terminal

**SYNOPSIS**

*ct* [ **-h** ] [ **-v** ] [ **-wn** ] [ **-sspeed** ] *telno* ...

**DESCRIPTION**

*Ct* dials the phone number of a modem that is attached to a terminal, and spawns a *getty* process to that terminal. *Telno* is a telephone number, with equal signs for secondary dial tones and minus signs for delays at appropriate places. If more than one telephone number is specified, *ct* will try each in succession until one answers; this is useful for specifying alternate dialing paths.

*Ct* will try each line listed in the file */usr/lib/uucp/L-devices* until it finds an available line with appropriate attributes or runs out of entries. If there are no free lines, *ct* will ask if it should wait for one, and if so, for how many minutes it should wait before it gives up. *Ct* will continue to try to open the dialers at one-minute intervals until the specified limit is exceeded. The dialogue may be overridden by specifying the **-wn** option, where *n* is the maximum number of minutes that *ct* is to wait for a line.

Normally, *ct* will hang up the current line, so that that line can answer the incoming call. The **-h** option will prevent this action. If the **-v** option is used, *ct* will send a running narrative to the standard error output stream.

The data rate may be set with the **-s** option, where *speed* is expressed in baud. The default rate is 300.

After the user on the destination terminal logs out, *ct* prompts, **Reconnect?** If the response begins with the letter **n** the line will be dropped; otherwise, *getty* will be started again and the **login:** prompt will be printed.

Of course, the destination terminal must be attached to a modem that can answer the telephone.

**FILES**

*/usr/lib/uucp/L-devices*  
*/usr/adm/ctlog*

**SEE ALSO**

*cu*(1C), *login*(1), *uucp*(1C).

## NAME

cu — call another UNIX system

## SYNOPSIS

cu [-sspeed] [-lline] [-h] [-t] [-d] [-m] [-o|e] telno | dir

## DESCRIPTION

*Cu* calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of ASCII files. *Speed* gives the transmission speed (110, 150, 300, 600, 1200, 4800, 9600); 300 is the default value. Most of our modems are either 300 or 1200 baud. For dial out lines, *cu* will choose a modem speed (300 or 1200) as the slowest available which will handle the specified transmission speed. Directly connected lines may be set to speeds higher than 1200 baud.

The *-l* value may be used to specify a device name for the communications line device to be used. This can be used to override searching for the first available line having the right speed. The speed of a line is taken from the file */usr/lib/uucp/L-devices*, overriding any speed specified by the *-s* option. The *-h* option emulates local echo, supporting calls to other computer systems which expect terminals to be in half-duplex mode. The *-t* option is used when dialing an ASCII terminal which has been set to auto-answer. Appropriate mapping of carriage-returns to carriage-return-line-feed pairs is set. The *-d* option cause diagnostic traces to be printed. The *-m* option specifies a direct line which has modem control. The *-o* (*-e*) option designates that even (odd) parity is to be generated for data sent to the remote. The *-d* option causes diagnostic traces to be printed. *Telno* is the telephone number, with equal signs for secondary dial tone or minus signs for delays, at appropriate places. The string *dir* for *telno* may be used for directly connected lines, and implies a null ACU. Using *dir* insures that a line has been specified by the *-l* option.

*Cu* will try each line listed in the file */usr/lib/uucp/L-devices* until it finds an available line with appropriate attributes or runs out of entries. 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 DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with *~* have special meanings.

The *transmit* process interprets the following:

- ~*. terminate the conversation.
- ~!* escape to an interactive shell on the local system.
- ~!cmd...* run *cmd* on the local system (via *sh -c*).
- ~\$cmd...* run *cmd* locally and send its output to the remote system.
- ~%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.

~... send the line ~... to the remote system.

~%nostop turn off the DC3/DC1 input control protocol for the remainder of the session. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters,

The *receive* process normally copies data from the remote system to its standard output. A line from the remote that begins with ~> initiates an output diversion to a file. The complete sequence is:

```
~> [>]: file
zero or more lines to be written to file
~>
```

Data from the remote is diverted (or appended, if >> is used) to file. The trailing ~> terminates the diversion.

The use of ~%put requires *stty(1)* and *cat(1)* on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of ~%take requires the existence of *echo(1)* and *cat(1)* on the remote system. Also, *stty tabs* mode should be set on the remote system if tabs are to be copied without expansion.

#### FILES

```
/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..(tty-device)
/dev/null
```

#### SEE ALSO

*cat(1)*, *ct(1C)*, *echo(1)*, *stty(1)*, *uucp(1C)*.

#### DIAGNOSTICS

Exit code is zero for normal exit, non-zero (various values) otherwise.

#### BUGS

*Cu* buffers input internally.

There is an artificial slowing of transmission by *cu* during the ~%put operation so that loss of data is unlikely.

**NAME**

**cut** — cut out selected fields of each line of a file

**SYNOPSIS**

**cut** -*clist* [*file1 file2 ...*]

**cut** -*flist* [-*dchar*] [-*s*] [*file1 file2 ...*]

**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 fixed length, i.e., character positions as on a punched card (-*c* option), or the length can vary from line to line and be marked with a field delimiter character like *tab* (-*f* option). *Cut* can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

- list*      A comma-separated list of integer field numbers (in increasing order), with optional - to indicate ranges as in the -*o* option of *nroff/troff* for page ranges; e.g., 1,4,7; 1-3,8; -5,10 (short for 1-5,10); or 3- (short for third through last field).
- clist*     The *list* following -*c* (no space) specifies character positions (e.g., -*c1-72* would pass the first 72 characters of each line).
- flist*     The *list* following -*f* is a list of fields assumed to be separated in the file by a delimiter character (see -*d* ); e.g. , -*f1,7* copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless -*s* is specified.
- dchar*    The character following -*d* is the field delimiter (-*f* option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.
- s*         Suppresses lines with no delimiter characters in case of -*f* option. Unless specified, lines with no delimiters will be passed through untouched.

Either the -*c* or -*f* option must be specified.

**HINTS**

Use *grep(1)* to make horizontal "cuts" (by context) through a file, or *paste(1)* to put files together column-wise (i.e., horizontally). To reorder columns in a table, use *cut* and *paste*.

**EXAMPLES**

**cut -d: -f1,5 /etc/passwd**

mapping of user IDs to names

**name=`who am i |cut -f1 -d" "`**

to set **name** to current login name.

**DIAGNOSTICS**

*line too long*

A line can have no more than 511 characters or fields.

*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.

*no fields*

The *list* is empty.

**SEE ALSO**

*grep(1)*, *paste(1)*.

**NAME**

*cw*, *checkcw* — prepare constant-width text for *troff*

**SYNOPSIS**

```
cw [ -lxx ] [ -rxx ] [ -fn ] [ -t ] [ +t ] [ -d ] [ files ]
checkcw [ -lxx ] [ -rxx ] files
```

**DESCRIPTION**

*Cw* is a preprocessor for *troff*(1) input files that contain text to be typeset in the constant-width (CW) font.

Text typeset with the CW font resembles the output of terminals and of line printers. This font is used to typeset examples of programs and of computer output in user manuals, programming texts, etc. (An earlier version of this font was used in typesetting *The C Programming Language* by B. W. Kernighan and D. M. Ritchie.) It has been designed to be quite distinctive (but not overly obtrusive) when used together with the Times Roman font.

Because the CW font contains a “non-standard” set of characters and because text typeset with it requires different character and inter-word spacing than is used for “standard” fonts, documents that use the CW font must be preprocessed by *cw*.

The CW font contains the 94 printing ASCII characters:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!%&'()*+@,./:;=?[ ]|-_`^~"<>{}#\
```

plus eight non-ASCII characters represented by four-character *troff*(1) names (in some cases attaching these names to “non-standard” graphics):

| <i>Character</i>        | <i>Symbol</i> | <i>Troff Name</i> |
|-------------------------|---------------|-------------------|
| “Cents” sign            | ¢             | \(ct              |
| EBCDIC “not” sign       | ~             | \(no              |
| Left arrow              | ←             | \(<-              |
| Right arrow             | →             | \(>-              |
| Down arrow              | ↓             | \(da              |
| Vertical single quote   | ’             | \(fm              |
| Control-shift indicator | ^             | \(dg              |
| Visible space indicator | ␣             | \(sq              |
| Hyphen                  | -             | \(hy              |

The hyphen is a synonym for the unadorned minus sign (-). Certain versions of *cw* recognize two additional names: \(\ua for an up arrow and \(\lh for a diagonal left-up (home) arrow.

*Cw* recognizes five request lines, as well as user-defined delimiters. The request lines look like *troff*(1) macro requests, and are copied in their entirety by *cw* onto its output; thus, they can be defined *by the user* as *troff*(1) macros; in fact, the .CW and .CN macros *should* be so defined (see *HINTS* below). The five requests are:

- .CW Start of text to be set in the CW font; .CW causes a break; it can take precisely the same options, in precisely the same format, as are available on the *cw* command line.
- .CN End of text to be set in the CW font; .CN causes a break; it can take the same options as are available on the *cw* command line.
- .CD Change delimiters and/or settings of other options; takes the same options as are available on the *cw* command line.

**.CP arg1 arg2 arg3 . . . argn**

All the arguments (which are delimited like *troff*(1) macro arguments) are concatenated, with the odd-numbered arguments set in the CW font and the even-numbered ones in the prevailing font.

**.PC arg1 arg2 arg3 . . . argn**

Same as .CP, except that the even-numbered arguments are set in the CW font and the odd-numbered ones in the prevailing font.

The .CW and .CN requests are meant to bracket text (e.g., a program fragment) that is to be typeset in the CW font "as is." Normally, *cw* operates in the *transparent* mode. In that mode, except for the .CD request and the nine special four-character names listed in the table above, every character between .CW and .CN request lines stands for itself. In particular, *cw* arranges for periods (.) and apostrophes (') at the beginning of lines, and backslashes (\) everywhere to be "hidden" from *troff*(1). The transparent mode can be turned off (see below), in which case normal *troff*(1) rules apply; in particular, lines that begin with . and ' are passed through untouched (except if they contain delimiters—see below). In either case, *cw* hides the effect of the font changes generated by the .CW and .CN requests; *cw* also defeats all ligatures (fi, ff, etc.) in the CW font.

The only purpose of the .CD request is to allow the changing of various options other than just at the beginning of a document.

The user can also define *delimiters*. The left and right delimiters perform the same function as the .CW/.CN requests; they are meant, however, to enclose CW "words" or "phrases" in running text (see example under *BUGS* below). *Cw* treats text between delimiters in the same manner as text enclosed by .CW/.CN pairs, except that, for aesthetic reasons, spaces and backspaces inside .CW/.CN pairs have the same width as other CW characters, while spaces and backspaces between delimiters are half as wide, so they have the same width as spaces in the prevailing text (but are *not* adjustable). Font changes due to delimiters are *not* hidden.

Delimiters have no special meaning inside .CW/.CN pairs.

The options are:

- lxx The one- or two-character string *xx* becomes the left delimiter; if *xx* is omitted, the left delimiter becomes undefined, which it is initially.
- rxx Same for the right delimiter. The left and right delimiters may (but need not) be different.
- fn The CW font is mounted in font position *n*; acceptable values for *n* are 1, 2, and 3 (default is 3, replacing the bold font). This option is only useful at the beginning of a document.
- t Turn transparent mode *off*.
- +t Turn transparent mode *on* (this is the initial default).
- d Print current option settings on file descriptor 2 in the form of *troff*(1) comment lines. This option is meant for debugging.

*Cw* reads the standard input when no *files* are specified (or when - is specified as the last argument), so it can be used as a filter. Typical usage is:

```
cw files | troff ...
```

*Checkcw* checks that left and right delimiters, as well as the .CW/.CN pairs, are properly balanced. It prints out all offending lines.



## HINTS

Typical definitions of the .CW and .CN macros meant to be used with the *mm(5)* macro package:

```
.de CW
.DS I
.ps 9
.vs 10.5p
.ta 16m/3u 32m/3u 48m/3u 64m/3u 80m/3u 96m/3u ...
..
.de CN
.ta 0.5i 1i 1.5i 2i 2.5i 3i 3.5i 4i 4.5i 5i 5.5i 6i
.vs
.ps
.DE
..
```

At the very least, the .CW macro should invoke the *troff(1)* no-fill (.nf) mode.

When set in running text, the CW font is meant to be set in the same point size as the rest of the text. In displayed matter, on the other hand, it can often be profitably set one point *smaller* than the prevailing point size (the displayed definitions of .CW and .CN above are one point smaller than the running text on this page). The CW font is sized so that, when it is set in 9-point, there are 12 characters per inch.

Documents that contain CW text may also contain tables and/or equations. If this is the case, the order of preprocessing should be: *cw*, *tbl*, and *eqn*. Usually, the tables contained in such documents will not contain any CW text, although it is entirely possible to have *elements* of the table set in the CW font; of course, care must be taken that *tbl(1)* format information not be modified by *cw*. Attempts to set equations in the CW font are not likely to be either pleasing or successful.

In the CW font, overstriking is most easily accomplished with backspaces: letting + represent a backspace,  $d++\backslash(d.g$  yields  $\hat{d}$ . (Because backspaces are half as wide between delimiters as inside .CW/.CN pairs—see above—two backspaces are required for each overstrike between delimiters.)

## FILES

/usr/lib/font/ftCW CW font-width table

## SEE ALSO

*eqn(1)*, *mmt(1)*, *tbl(1)*, *troff(1)*, *mm(5)*, *mv(5)*.

## WARNINGS

If text preprocessed by *cw* is to make any sense, it must be set on a typesetter equipped with the CW font or on a STARE facility; on the latter, the CW font appears as bold, but with the proper CW spacing.

## BUGS

Only a masochist would use periods (.), backslashes (\), or double quotes (") as delimiters, or as arguments to .CP and .PC.

Certain CW characters don't concatenate gracefully with certain Times Roman characters, e.g., a CW ampersand (&) followed by a Times Roman comma (,); in such cases, judicious use of *troff(1)* half- and quarter-spaces (\i and \^ ) is most salutary, e.g., one should use  $\& \backslash^$ , (rather than just plain  $\& ,$ ) to obtain  $\&$ , (assuming that  $\_$  is used for both delimiters). Using *cw* with *nroff* is silly.

The output of *cw* is hard to read.

See also *BUGS* under *troff(1)*.

**NAME**

`cxref` — generate C program cross reference

**SYNOPSIS**

`cxref` [ options ] files

**DESCRIPTION**

*Cxref* analyzes a collection of C files and attempts to build a cross reference table. *Cxref* utilizes a special version of *cpp* 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.

In addition to the `-D`, `-I` and `-U` options (which are identical to their interpretation by *cc*(1)), 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 file` Direct output to named *file*.
- `-s` Operate silently; does not print input file names.
- `-t` Format listing for 80-column width.

**FILES**

`/usr/lib/xcpp` special version of C-preprocessor.

**SEE ALSO**

*cc*(1).

**DIAGNOSTICS**

Error messages are unusually cryptic, but usually mean that you can't compile these files, anyway.

**NAME**

date — print and set the date

**SYNOPSIS**

date [ mmddhhmm[yy] ] [ +format ]

**DESCRIPTION**

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. 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. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time.

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*(3S). 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

```

**EXAMPLE**

```

date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
would have generated as output:
DATE: 08/01/76
TIME: 14:45:05

```

**DIAGNOSTICS**

*No permission* if you aren't the super-user and you try to change the date;

*bad conversion* if the date set is syntactically incorrect;

*bad format character* if the field descriptor is not recognizable.

**FILES**

/dev/kmem

**WARNING**

It is a bad practice to change the date while the system is running multi-user.

## NAME

dc — desk calculator

## SYNOPSIS

dc [ file ]

## DESCRIPTION

*Dc* is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. 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, then from the standard input. The following constructions are recognized:

*number*

The value of the number is pushed on 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. Numbers 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; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

`sx` The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the *s* is capitalized, *x* is treated as a stack and the value is pushed on it.

`lx` The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the *l* is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

`d` The top value on the stack is duplicated.

`p` The top value on the stack is printed. The top value remains unchanged. `P` interprets the top of the stack as an ASCII string, removes it, and prints it.

`f` All values on the stack are printed.

`q` exits the program. If executing a string, the recursion level is popped by two. If `q` is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

`x` treats the top element of the stack as a character string and executes it as a string of *dc* commands.

`X` replaces the number on the top of the stack with its scale factor.

`[ ... ]` puts the bracketed ASCII string onto the top of the stack.

`<x >x =x`

The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.

`v` replaces 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.

`!` interprets the rest of the line as a UNIX command.

`c` All values on the stack are popped.

- i The top value on the stack is popped and used as the number radix for further input. I pushes the input base on the top of the stack.
- o The top value on the stack is popped and used as the number radix for further output.
- O pushes 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 The stack level is pushed onto the stack.
- Z replaces the number on the top of the stack with its length.
- ? A line of input is taken from the input source (usually the terminal) and executed.
- ; : are used by *bc* for array operations.

**EXAMPLE**

This example prints the first ten values of *n!*:

```
[|a| + dsa*pla10>y]sy
0sa1
lyx
```

**SEE ALSO**

*bc(1)*, which is a preprocessor for *dc* providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

**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.

## NAME

**dd** — convert and copy a file

## SYNOPSIS

**dd** [option=value] ...

## DESCRIPTION

*Dd* copies the specified input file to the 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.

| <i>option</i>              | <i>values</i>                                                                                                                                                                      |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>if</b> = <i>file</i>    | input file name; standard input is default                                                                                                                                         |
| <b>of</b> = <i>file</i>    | output file name; standard output is default                                                                                                                                       |
| <b>ibs</b> = <i>n</i>      | input block size <i>n</i> bytes (default 512)                                                                                                                                      |
| <b>obs</b> = <i>n</i>      | output block size (default 512)                                                                                                                                                    |
| <b>bs</b> = <i>n</i>       | set both input and output block size, superseding <i>ibs</i> and <i>obs</i> ; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done |
| <b>cbs</b> = <i>n</i>      | conversion buffer size                                                                                                                                                             |
| <b>skip</b> = <i>n</i>     | skip <i>n</i> input records before starting copy                                                                                                                                   |
| <b>seek</b> = <i>n</i>     | seek <i>n</i> records from beginning of output file before copying                                                                                                                 |
| <b>count</b> = <i>n</i>    | copy only <i>n</i> input records                                                                                                                                                   |
| <b>conv</b> = <b>ascii</b> | convert EBCDIC to ASCII                                                                                                                                                            |
| <b>ebcdic</b>              | convert ASCII to EBCDIC                                                                                                                                                            |
| <b>ibm</b>                 | slightly different map of ASCII to EBCDIC                                                                                                                                          |
| <b>lcase</b>               | map alphabetic to lower case                                                                                                                                                       |
| <b>ucase</b>               | map alphabetic to upper case                                                                                                                                                       |
| <b>swab</b>                | swap every pair of bytes                                                                                                                                                           |
| <b>noerror</b>             | do not stop processing on an error                                                                                                                                                 |
| <b>sync</b>                | pad every input record to <i>ibs</i>                                                                                                                                               |
| ..., ...                   | several comma-separated conversions                                                                                                                                                |

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by **x** to indicate a product.

*Cbs* is used only if *ascii* or *ebcdic* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, 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.

## EXAMPLE

This command will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file *x*:

```
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.

## SEE ALSO

cp(1).

**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.

New-lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

## NAME

delta — make a delta (change) to an SCCS file

## SYNOPSIS

delta [-rSID] [-s] [-n] [-glist] [-m[mrlist]] [-y[comment]] [-p] files

## DESCRIPTION

*Delta* is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get(1)* (called the *g-file*, or generated file).

*Delta* makes a delta to each named SCCS file. If a directory is named, *delta* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s.*) and unreadable files are silently ignored. If a name of *-i* is given, the standard input is read (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

*Delta* may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin(1)*) that may be present in the SCCS file (see *-m* and *-y* keyletters below).

Keyletter arguments apply independently to each named file.

- rSID           Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *gets* for editing (*get -e*) on the same SCCS file were done by the same person (login name). The SID value specified with the *-r* keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get(1)*). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.
- s              Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.
- n              Specifies retention of the edited *g-file* (normally removed at completion of delta processing).
- glist          Specifies a *list* (see *get(1)*) for the definition of *list* of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.
- m[mrlist]     If the SCCS file has the *v* flag set (see *admin(1)*) then a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.  
  
If *-m* is not used and the standard input is a terminal, the prompt *MRs?* is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The *MRs?* prompt always precedes the *comments?* prompt (see *-y* keyletter).  
  
MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.



Note that if the *v* flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from MR number validation program, *delta* terminates (it is assumed that the MR numbers were not all valid).

**-y**[*comment*] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If *-y* is not specified and the standard input is a terminal, the prompt *comments?* is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

**-p** Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

## FILES

All files of the form *?-file* are explained in the *Source Code Control System User's Guide*. The naming convention for these files is also described there.

**g-file** Existed before the execution of *delta*; removed after completion of *delta*.

**p-file** Existed before the execution of *delta*; may exist after completion of *delta*.

**q-file** Created during the execution of *delta*; removed after completion of *delta*.

**x-file** Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.

**z-file** Created during the execution of *delta*; removed during the execution of *delta*.

**d-file** Created during the execution of *delta*; removed after completion of *delta*.

**/usr/bin/bdiff** Program to compute differences between the "gotten" file and the *g-file*.

## WARNINGS

Lines beginning with an SOH ASCII 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 will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get/delta* sequences should be used.

If the standard input (*-*) is specified on the *delta* command line, the *-m* (if necessary) and *-y* keyletters *must* also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

## SEE ALSO

*admin*(1), *bdiff*(1), *cdc*(1), *get*(1), *help*(1), *prs*(1), *rmdel*(1), *sccsfile*(4).  
*Source Code Control System User's Guide* in the *UNIX System User's Guide*.

## DIAGNOSTICS

Use *help*(1) for explanations.

**NAME**

**deroff** — remove nroff/troff, tbl, and eqn constructs

**SYNOPSIS**

**deroff** [ **-mx** ] [ **-w** ] [ files ]

**DESCRIPTION**

*Deroff* reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between .EQ and .EN lines, and between delimiters), and *tbl*(1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. *Deroff* follows chains of included files (.so and .nx *troff* commands); if a file has already been included, a .so naming that file is ignored and a .mx naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The **-m** option may be followed by an **m**, **s**, or **l**. The **-mm** option causes the macros be interpreted so that only running text is output (i.e., no text from macro lines.) The **-ml** option forces the **-mm** option and also causes deletion of lists associated with the **mm** macros.

If the **-w** option is given, the output is a word list, one “word” per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a “word” is any string that *contains* at least two letters and is composed of letters, digits, ampersands (&), and apostrophes ('); in a macro call, however, a “word” is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from “words.”

**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 such errors result in too much rather than too little output.

The **-ml** option does not handle nested lists correctly.



## NAME

diff - differential file comparator

## SYNOPSIS

diff [ -efbh ] file1 file2

## DESCRIPTION

*Diff* tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is -, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

```
n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4
```

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging *a* for *d* and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, 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 >.

The -b option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The -e option produces a script of *a*, *c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The -f option produces a similar script, not useful with *ed*, in the opposite order. 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
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option -h does 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 -e and -f are unavailable with -h.

## FILES

```
/tmp/d????
/usr/lib/diffh for -h
```

## SEE ALSO

cmp(1), comm(1), ed(1).

## DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

## BUGS

Editing scripts produced under the -e or -f option are naive about creating lines consisting of a single period (.).

**NAME**

diff3 - 3-way differential file comparison

**SYNOPSIS**

diff3 [ -ex3 ] file1 file2 file3

**DESCRIPTION**

*Diff3* compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

- ==== all three files differ
- ====1 *file1* is different
- ====2 *file2* is different
- ====3 *file3* is different

The type of change suffered in converting a given range of a given file to some other is 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.

Under the *-e* option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged *====* and *=====3*. Option *-x (-3)* produces a script to incorporate only changes flagged *==== (======3)*. The following command will apply the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed - file1
```

**FILES**

- /tmp/d3\*
- /usr/lib/diff3prog

**SEE ALSO**

diff(1).

**BUGS**

Text lines that consist of a single . will defeat *-e*.  
Files longer than 64K bytes won't work.



**NAME**

`diffmk` — mark differences between files

**SYNOPSIS**

`diffmk` *name1* *name2* *name3*

**DESCRIPTION**

*Diffmk* compares two versions of a file and creates a third file that includes “change mark” commands for *nroff* or *troff*(1). *Name1* and *name2* are the old and new versions of the file. *Diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter “change mark” (.mc) requests. When *name3* 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 \*.

If anyone is so inclined, *diffmk* can be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file *macs* contains:

```
.pl 1
.ll 77
.nf
.eo
.nc
```

The *.ll* request might specify a different line length, depending on the nature of the program being printed. The *.eo* and *.nc* requests are probably needed only for C programs.

If the characters | and \* are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shell procedure).

**SEE ALSO**

`diff`(1), `nroff`(1), `troff`(1).

**BUGS**

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing *.sp* by *.sp 2* will produce a “change mark” on the preceding or following line of output.

**NAME**

`dircmp` — directory comparison

**SYNOPSIS**

`dircmp` [ `-d` ] [ `-s` ] `dir1 dir2`

**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.

- `-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.

**SEE ALSO**

`cmp`(1), `diff`(1).

**NAME**

dis - 3B20S disassembler

**SYNOPSIS**

dis [-o] [-V] [-L] [-d sec] [-da sec] [-t sec] [-l string] files

**DESCRIPTION**

The *dis* command produces an assembly language listing of each of its object *file* arguments. The listing includes assembly statements and the binary that produced those statements.

The following *options* are interpreted by the disassembler and may be specified in any order.

- o Will print numbers in octal. Default is hexadecimal.
- V Version number of the disassembler will be written to standard error.
- L Invokes a lookup of C source labels in the symbol table for subsequent printing.
- d sec Disassembles the named section as data, printing the offset of the data from the beginning of the section.
- da sec Disassembles the named section as data, printing the actual address of the data.
- t sec Disassembles the named section as text.
- l string Will disassemble the library file specified as *string*. For example, one would issue the command `dis -l x -l z` to disassemble `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 C breakpointable line number, starts with the following instruction. An expression such as <40> in the operand field, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A C function name will appear in the first column, followed by ( ).

**SEE ALSO**

as(1), cc(1), ld(1).

**DIAGNOSTICS**

The self explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

## NAME

*dpd*, *lpd* — HONEYWELL sending daemon, line printer daemon

## SYNOPSIS

*/usr/lib/dpd*  
*/usr/lib/lpd*

## DESCRIPTION

*Dpd* is the daemon for the 200-series DATA-PHONE® set or for a KMC11-B using *vpm(7)*. It is designed to submit jobs to the HONEYWELL 6000 computer via the GRTS interface. *Lpd* is the daemon for a line printer.

*Dpd* uses the directory */usr/spool/dpd*. *Lpd* uses the directory */usr/spool/lpd*. The file lock in either directory is used to prevent two daemons from becoming active simultaneously. After the program has successfully set the lock, it forks and the main path exits, thus spawning the daemon. The directory is scanned for files beginning with "df". Each such file is submitted as a job. Each line of a job file must begin with a key character to specify what to do with the remainder of the line.

- S** directs *dpd* to generate a unique *snumb card*. The *snumb* number is generated from the file *snumb* in the spooling directory in the case of the DATA-PHONE set daemon. This key character is not used by *lpd*.
- L** specifies that the remainder of the line is to be sent as a literal.
- I** is the same as **L**, but signals the \$ IDENT card which is to be mailed back by the mail option.
- B** specifies that the rest of the line is a file name. That file is to be sent as binary cards.
- F** is the same as **B** except a form-feed is prepended to the file.
- U** specifies that the rest of the line is a file name. After the job has been transmitted, the file is unlinked.
- M** is followed by a user ID; after the job is sent, a message is mailed to the user via the *mail(1)* command to verify the sending of the job.
- N** is followed by a user file name, to be sent back under the mail option.
- Q** is followed by a string of characters, which is a message to be sent back to the user under the mail option. (Not used by *lpd*).

Any error encountered will cause the daemon to drop the call, wait up to 20 minutes, (only 10 seconds for *lpd*), and start over. This means that an improperly constructed "df" file may cause the same job to be submitted every 20 minutes.

*Dpd* is automatically initiated by all of the GCOS commands (*dpr*, *gcat*, *gcosmail*, *fget*, and *fsend*). *Lpd* is automatically initiated by the line printer command, *lpr*.

To restart *dpd* or *lpd* (in the case of hardware or software malfunction), it is necessary to first kill the old daemon (if it is still alive), and remove the lock file (if present), before initiating the new daemon. This can be done automatically by */etc/rc* when the system is brought up, in the event there were jobs left in the spooling directory when the system last went down.

## FILES

|                         |                                     |
|-------------------------|-------------------------------------|
| <i>/usr/spool/dpd/*</i> | spool area for GCOS daemons.        |
| <i>/usr/spool/lpd/*</i> | spool area for line printer daemon. |
| <i>/etc/passwd</i>      | to get the user's name.             |
| <i>/dev/dn?</i>         | ACU device.                         |
| <i>/dev/du?</i>         | DATA-PHONE set.                     |



/dev/vpm?           VPM device to interface to KMC11-B.  
/dev/lp             line printer device.

**SEE ALSO**

dpr(1C), fget(1C), fsend(1C), gcat(1C), gcosmail(1C), lpr(1).

**BUGS**

If a *umask*(1) of 077 is used, the print jobs may be spooled but won't be able to be printed.

## NAME

dpr — off-line print

## SYNOPSIS

**dpr** [ *--destination* ] [ *options* ] [ *files* ]

## DESCRIPTION

*Dpr* causes the named files to be printed off-line at the specified destination, by GCOS at the Murray Hill Computation Center. GCOS identification must appear in the UNIX password file (see *passwd(4)*), or be supplied by the *-i* option. If no files are listed the standard input is assumed; thus *dpr* may be used as a filter.

The destination is a two-character code which is taken to be a Murray Hill GCOS "station id." Useful codes are *r1* for quality print, and *q1* for quality print with special ribbon, both on regular wide paper. The codes *r2* and *q2* give the same print on narrow paper. The code *mx* is a Xerox 9700 printer. The default destination is on-line at the Murray Hill Computation Center.

The following options, each as a separate argument, and in any combination (multiple outputs are permitted), may be given before or after the destination:

- c* Makes a copy of the file to be sent before returning to the user.
- r* Removes the file after sending it.
- f file* Use *file* as a dummy file name to report back in the mail. (This is useful for distinguishing multiple runs, especially when *dpr* is being used as a filter).
- i job,bin*  
Supply the GCOS "ident card" image as the parameter *-i job,bin* where *job* is the GCOS job number and *bin* the GCOS bin number or any comment to the GCOS operators.
- m* When transmission is complete, reports by *mail(1)* the so-called *snumb* of the receiving GCOS job. The mail is sent by the UNIX daemon; there is no guarantee that the GCOS job ran successfully. This is the default option.
- n* Does not report the completion of transmission by *mail(1)*.
- p* Selects portrait mode. Used in conjunction with a XEROX 9700 printer.
- sn* Submits job to GCOS with service grade *n* (*n*=1, 2, 3, 4). Default is *-s2*.

## EXAMPLES

The command:

```
dpr -r -n error1 error2
```

will send the files *error1* and *error2* to GCOS for printing, removing the files after they have been sent, but not sending mail. The line:

```
pr file1 | dpr -s1 -fjob1 -r1
```

will send the output of *pr* to GCOS for printing on the quality printer with service grade 1, and will send mail that *job1* has been sent.

## FILES

|                         |                                            |
|-------------------------|--------------------------------------------|
| <i>/etc/passwd</i>      | user's identification and GCOS ident card. |
| <i>/usr/lib/dpd</i>     | sending daemon.                            |
| <i>/usr/spool/dpd/*</i> | spool area.                                |

## SEE ALSO

*dpd(1C)*, *fget(1C)*, *fsend(1C)*, *gcat(1C)*.

**NAME**

**du** — summarize disk usage

**SYNOPSIS**

**du** [ **-ars** ] [ **names** ]

**DESCRIPTION**

*Du* gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, . is used.

The optional argument **-s** causes only the grand total (for each of the specified *names*) to be given. The optional argument **-a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

*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.

A file with two or more links is only counted once.

**BUGS**

If the **-a** option is not used, non-directories given as arguments are not listed.

If there are too many distinct linked files, *du* will count the excess files more than once.

Files with holes in them will get an incorrect block count.

**NAME**

dump — dump selected parts of an object file

**SYNOPSIS**

dump [-a] [-f] [-o] [-h] [-s] [-r] [-l] [-t] [-z name] files

**DESCRIPTION**

The *dump* command dumps selected parts of each of its object *file* arguments.

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:

- a           Dump the archive header of each member of each archive file argument.
- 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.

The following *modifiers* are used in conjunction with the options listed above to modify their capabilities.

- d number    Dump the section number or range of sections starting at *number* and ending either at the last section number or *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.
- 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.
- 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 options of *dump*.
- 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.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the -z option may be replaced by a blank.

The *dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

**SEE ALSO**

✓ a.out(4), ar(4).

**NAME**

echo — echo arguments

**SYNOPSIS**

echo [ arg ] ...

**DESCRIPTION**

*Echo* writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

|                 |                                                                                                                     |
|-----------------|---------------------------------------------------------------------------------------------------------------------|
| <code>\b</code> | backspace                                                                                                           |
| <code>\c</code> | print line without new-line                                                                                         |
| <code>\f</code> | form-feed                                                                                                           |
| <code>\n</code> | new-line                                                                                                            |
| <code>\r</code> | carriage return                                                                                                     |
| <code>\t</code> | tab                                                                                                                 |
| <code>\\</code> | backslash                                                                                                           |
| <code>\n</code> | the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number <i>n</i> , which must start with a zero. |

*Echo* is useful for producing diagnostics in command files and for sending known data into a pipe.

**SEE ALSO**

sh(1).

1

## NAME

ed, red - text editor

## SYNOPSIS

ed [ - ] [ -x ] [ file ]

red [ - ] [ -x ] [ file ]

## DESCRIPTION

*Ed* is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. The optional *-* suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the *!* prompt after a *!shell command*. If *-x* is present, an *x* command is simulated first to handle an encrypted file. *Ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

*Red* is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via *!shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec(4)* formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in *stty -tabs* or *stty tab3* mode (see *stty(1)*), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10 and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

*Ed* supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character REs* match a *single* character:

- 1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.
- 1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

- a. ., \*, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([ ]); see 1.4 below).
  - b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([ ]) (see 1.4 below).
  - c. \$ (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).
  - d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)
- 1.3 A period (.) is a one-character RE that matches any character except new-line.
- 1.4 A non-empty string of characters enclosed in square brackets ([ ]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line 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); e.g., []a-f] matches either a right square bracket (]) or one of the letters a through f inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct REs from one-character REs:

- 2.1 A one-character RE is a RE that matches whatever the one-character RE matches.
- 2.2 A one-character RE followed by an asterisk (\*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.
- 2.3 A one-character RE followed by \{m\}, \{m,\}, or \{m,n\} is a RE that matches a *range* of occurrences of the one-character RE. 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 RE matches as many occurrences as possible.
- 2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.
- 2.5 A RE enclosed between the character sequences \( and\) is a RE that matches whatever the unadorned RE matches.
- 2.6 The expression \n matches the same string of characters as was matched by an expression enclosed between \( and\) *earlier* in the same RE. Here *n* is a digit; the sub-expression 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.



Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both):

- 3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.
- 3.2 A currency symbol (\$) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction  $\wedge$ entire RE\$ constrains the entire RE to match the entire line.

The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character . addresses the current line.
2. The character \$ addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. 'x addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; e.g. -5 is understood to mean .-5.
9. If an address ends with + or -, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address - refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to -.) Moreover, trailing + and - characters have a cumulative effect, so -- refers to the current line less 2.
10. For convenience, a comma (,) stands for the address pair 1,\$, while a semicolon (;) stands for the pair .,\$.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

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 suffixed by *l*, *n* or *p*, in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n* and *p* commands.

(.)**a**  
<text>

The *append* command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(.)**c**  
<text>

The *change* command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(...)**d**

The *delete* command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

**e file**

The *edit* command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

**E file**

The *Edit* command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

**f** *file*

If *file* is given, the *file-name* command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

**(1,\$)g**/*RE*/*command list*

In the global command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with *.* initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a *\*; *a*, *i*, and *c* commands and associated input are permitted; the *.* terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

**(1,\$)G**/*RE*/

In the interactive Global command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, *.* is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an *&* causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

**h**

The *help* command gives a short error message that explains the reason for the most recent *?* diagnostic.

**H**

The *Help* command causes *ed* to enter a mode in which error messages are printed for all subsequent *?* diagnostics. It will also explain the previous *?* if there was one. The *H* command alternately turns this mode on and off; it is initially off.

**(.)i**

<text>

The *insert* command inserts the given text before the addressed line; *.* is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal 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**

The *join* command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

**(.)kx**

The *mark* command marks the addressed line with name *x*, which must be a lower-case letter. The address '*x*' then addresses this line; *.* is unchanged.

(...)**l**

The *list* command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab*, *backspace*) are represented by (hopefully) mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(...)**ma**

The *move* command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(...)**n**

The *number* command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(...)**p**

The *print* command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*; for example, *dp* deletes the current line and prints the new current line.

**P**

The editor will prompt with a \* for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

**q**

The *quit* command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

**Q**

The editor exits without checking if changes have been made in the buffer since the last *w* command.

(\$)**r file**

The *read* command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "\$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(...)**s/RE/replacement/** or(...)**s/RE/replacement/g**

The *substitute* command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator *g* appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the

substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters \n, where n is a digit, are replaced by the text matched by the n-th regular subexpression of the specified RE enclosed between \ ( and \). When nested parenthesized subexpressions are present, n is determined by counting occurrences of \ ( starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a g or v command list.

(.,.)*ta*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u

The *undo* command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(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/

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1,\$)*w* *file*

The write command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

X

A key string is demanded from the standard input. Subsequent *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

(**\$**)=

The line number of the addressed line is typed; . is unchanged by this command.

**!***shell command*

The remainder of the line after the ! is sent to the UNIX shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(**.+1**)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it 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.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. 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 new-line. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may 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?      |

**FILES**

/tmp/e# temporary; # is the process number.  
ed.hup work is saved here if the terminal is hung up.

**DIAGNOSTICS**

? for command errors.  
?file for an inaccessible file.  
(use the *help* and *Help* commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands: it prints ? and allows one to continue editing. A second *e* or *q* command at this point will take effect. The - command-line option inhibits this feature.

**SEE ALSO**

crypt(1), grep(1), sed(1), sh(1), stty(1), fspec(4), regexp(5).  
*A Tutorial Introduction to the UNIX Text Editor* by B. W. Kernighan.  
*Advanced Editing on UNIX* by B. W. Kernighan.

**CAVEATS AND BUGS**

A ! command cannot be subject to a *g* or a *v* command.  
The ! command and the ! escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).  
The sequence \n in a RE does not match a new-line character.  
The ! command mishandles DEL.

Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.  
Characters are masked to 7 bits on input.

**NAME**

**efl** — Extended Fortran Language

**SYNOPSIS**

**efl** [ options ] [ files ]

**DESCRIPTION**

*Efl* compiles a program written in the EFL language into clean Fortran on the standard output. *Efl* provides the C-like control constructs of *ratfor*(1):

statement grouping with braces.

decision-making:

**if**, **if-else**, and **select-case** (also known as **switch-case**);  
**while**, **for**, Fortran **do**, **repeat**, and **repeat ... until** loops;  
 multi-level **break** and **next**.

EFL has C-like data structures, e.g.:

```
struct
{
  integer flags(3)
  character(8) name
  long real coords(2)
  } table(100)
```

The language offers generic functions, assignment operators (**+=**, **&=**, etc.), and sequentially evaluated logical operators (**&&** and **||**). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) })
```

EFL also provides some syntactic "sugar":

free-form input:

multiple statements per line; automatic continuation; statement label names (not just numbers).

comments:

**#** this is a comment.

translation of relational and logical operators:

**>**, **>=**, **&**, etc., become **.GT.**, **.GE.**, **.AND.**, etc.

return expression to caller from function:

**return** (*expression*)

defines:

**define** *name replacement*

includes:

**include** *file*

*Efl* understands several option arguments: **-w** suppresses warning messages, **-#** suppresses comments in the generated program, and the default option **-C** causes comments to be included in the generated program.

An argument with an embedded **=** (equal sign) sets an EFL option as if it had appeared in an **option** statement at the start of the program. Many options are described in the reference manual. A set of defaults for a particular target machine may be selected by one of the choices: **system=unix**, **system=gcos**, or **system=cray**. The default setting of the **system** option is the same as the machine the compiler is running on. Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.



*Efl* is best used with *f77*(1).

**SEE ALSO**

*cc*(1), *f77*(1), *ratfor*(1).

*The Programming Language EFL* by S.I. Feldman.

**NAME**

enable, disable — enable/disable LP printers

**SYNOPSIS**

**enable** printers

**disable** [-c] [-r[reason]] printers

**DESCRIPTION**

*Enable* activates the named *printers*, enabling them to print requests taken by *lp(1)*. Use *lpstat(1)* to find the status of printers.

*Disable* deactivates the named *printers*, disabling them from printing requests taken by *lp(1)*. By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat(1)* to find the status of printers. Options useful with *disable* are:

- c Cancel any requests that are currently printing on any of the designated printers.
- r[reason] Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next -r option. If the -r option is not present or the -r option is given without a reason, then a default reason will be used. *Reason* is reported by *lpstat(1)*.

**FILES**

/usr/spool/lp/\*

**SEE ALSO**

*lp(1)*, *lpstat(1)*.

**NAME**

`env` - set environment for command execution

**SYNOPSIS**

`env [-] [ name=value ] ... [ command args ]`

**DESCRIPTION**

*Env* obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The `-` flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

**SEE ALSO**

`sh(1)`, `exec(2)`, `profile(4)`, `environ(5)`.

## NAME

eqn, neqn, checkeq — format mathematical text for nroff or troff

## SYNOPSIS

eqn [ -dxy ] [ -pn ] [ -sn ] [ -fn ] [ files ]

neqn [ -dxy ] [ -pn ] [ -sn ] [ -fn ] [ files ]

checkeq [ files ]

## DESCRIPTION

*Eqn* is a *troff*(1) preprocessor for typesetting mathematical text on a phototypesetter, while *neqn* is used for the same purpose with *nroff* on typewriter-like terminals. Usage is almost always:

```
eqn files | troff
neqn files | nroff
```

or equivalent.

If no files are specified (or if — is specified as the last argument), these programs read 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 designate two characters as *delimiters*; subsequent text between delimiters is then treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument *-dxy* or (more commonly) with *delim xy* between *.EQ* and *.EN*. The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by *delim off*. All text that is neither between delimiters nor between *.EQ* and *.EN* is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and *.EQ/.EN* pairs.

Tokens within *eqn* are separated by spaces, tabs, new-lines, braces, double quotes, tildes, and circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character such as *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 j* makes  $x_j$ , *a sub k sup 2* produces  $a_k^2$ , while  $e^{x^2+y^2}$  is made with *e sup {x sup 2 + y sup 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:  $\lim_{n \rightarrow \infty} \sum_0^n x_i$  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  $\left[ x^2 + \frac{y^2}{\alpha} \right] = 1$ .

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). A *left thing* need not have a matching *right thing*.

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**:  
*pile* { *a* above *b* above *c* } produces  $\begin{matrix} a \\ b \\ c \end{matrix}$ . Piles may have arbitrary numbers of  
 elements; **lpile** left-justifies, **pile** and **cpile** center (but with different verti-  
 cal 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  $\bar{x} = \overline{f(t)}$ , *y dotdot bar*  $\tilde{\sim} = \tilde{\sim} n$  *under* is  $\underline{y} = \underline{n}$ ,  
 and *x vec*  $\vec{\sim} = \vec{\sim} y$  *dyad* is  $\vec{x} = \vec{y}$ .

Point sizes and fonts can be changed with **size** *n* or **size**  $\pm n$ , **roman**, **italic**,  
**bold**, and **font** *n*. Point sizes and fonts can be changed globally in a docu-  
 ment by **gsize** *n* and **gfont** *n*, or by the command-line arguments **-sn** and  
**-fn**.

Normally, subscripts and superscripts are reduced by 3 points from the pre-  
 vious size; this may be changed by the command-line argument **-pn**.

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* that will be replaced by *replacement* when-  
 ever it appears thereafter. The % may be any character that does not occur  
 in *replacement*.

Keywords such as **sum** ( $\Sigma$ ), **int** ( $\int$ ), **inf** ( $\infty$ ), and shorthands such as  
 $\geq$  (**>=**),  $\neq$  (**!=**), and  $\rightarrow$  (**->**) are recognized. Greek letters are spelled  
 out in the desired case, as in **alpha** ( $\alpha$ ), or **GAMMA** ( $\Gamma$ ). Mathematical  
 words such as **sin**, **cos**, and **log** are made Roman automatically. **Troff**(1)  
 four-character escapes such as **\(dd** ( $\ddagger$ ) and **\(bs** ( $\textcircled{\text{B}}$ ) may be used any-  
 where. 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**(1) when all else fails. Full details are given in the  
 manual cited below.

#### SEE ALSO

*Typesetting Mathematics—User's Guide* by B. W. Kernighan and L. L.  
 Cherry.  
**cw**(1), **mm**(1), **mmt**(1), **nroff**(1), **tbl**(1), **troff**(1), **eqnchar**(5), **mm**(5),  
**mv**(5).

#### BUGS

To embolden digits, parentheses, etc., it is necessary to quote them, as in  
**bold "12.3"**.

See also **BUGS** under **troff**(1).

**NAME**

*expr* — evaluate arguments as an expression

**SYNOPSIS**

*expr* arguments

**DESCRIPTION**

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that 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, 2'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*

returns the first *expr* if it is neither null nor 0, otherwise returns the second *expr*.

*expr* \& *expr*

returns the first *expr* if neither *expr* is null or 0, otherwise returns 0.

*expr* { =, \>, \>=, \<, \<=, != } *expr*

returns 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.

*expr* : *expr*

The matching operator : compares 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" (i.e., 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.

**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 (i.e., file). 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).

#### EXIT CODE

As a side effect of expression evaluation, *expr* returns the following exit values:

|   |                                         |
|---|-----------------------------------------|
| 0 | if the expression is neither null nor 0 |
| 1 | if the expression is null or 0          |
| 2 | for invalid expressions.                |

#### DIAGNOSTICS

|                             |                                             |
|-----------------------------|---------------------------------------------|
| <i>syntax error</i>         | for operator/operand errors                 |
| <i>non-numeric argument</i> | if arithmetic is attempted on such a string |

#### 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=
```

## NAME

*f77* — Fortran 77 compiler

## SYNOPSIS

*f77* [ options ] files

## DESCRIPTION

*F77* is the UNIX Fortran 77 compiler; it accepts several types of *file* arguments:

Arguments whose names end with *.f* are taken to be Fortran 77 source programs; they are compiled and each object program is left in the current directory in a file whose name is that of the source, with *.o* substituted for *.f*.

Arguments whose names end with *.r* or *.e* are taken to be RATFOR or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by *f77*, producing *.o* files.

In the same way, arguments whose names end with *.c* or *.s* are taken to be C or assembly source programs and are compiled or assembled, producing *.o* files.

The following *options* have the same meaning as in *cc*(1) (see *ld*(1) for link editor options):

- c Suppress link editing and produce *.o* files for each source file.
- p Prepare object files for profiling (see *prof*(1)).
- O Invoke an object-code optimizer.
- S Compile the named programs and leave the assembler-language output in corresponding files whose names are suffixed with *.s*. (No *.o* files are created.)
- o*output* Name the final output file *output*, instead of *a.out*.
- f In systems without floating-point hardware, use a version of *f77* that handles floating-point constants and links the object program with the floating-point interpreter.
- g Generate additional information needed for the use of *sdb*(1) (VAX-11/780 only).

The following *options* are peculiar to *f77*:

- onetrip Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)
- 1 Same as -onetrip.
- 66 Suppress extensions which enhance Fortran 66 compatibility.
- C Generate code for run-time subscript range-checking.
- I[24s] Change the default size of integer variables (only valid on machines where the "normal" integer size is not equal to the size of a single precision real). -I2 causes all integers to be 2-byte quantities, -I4 (default) causes all integers to be 4-byte quantities, and -Is changes the default size of subscript expressions (only) from the size of an integer to 2 bytes.
- U Do not "fold" cases. *F77* is normally a no-case language (i.e. *a* is equal to *A*). The -U option causes *f77* to treat upper and lower cases to be separate.
- u Make the default type of a variable *undefined*, rather than using the default Fortran rules.
- w Suppress all warning messages. If the option is -w66, only Fortran 66 compatibility warnings are suppressed.



- F Apply EFL and RATFOR preprocessor to relevant files, put the result in files whose names have their suffix changed to `.of`. (No `.o` files are created.)
- m Apply the M4 preprocessor to each EFL or RATFOR source file before transforming with the `ratfor(1)` or `efl(1)` processors.
- E The remaining characters in the argument are used as an EFL flag argument whenever processing a `.e` file.
- R The remaining characters in the argument are used as a RATFOR flag argument whenever processing a `.r` file.

Other arguments are taken to be either link-editor option arguments or *f77*-compatible object programs (typically produced by an earlier run), or libraries of *f77*-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the default name `a.out`.

## FILES

|                                |                                          |
|--------------------------------|------------------------------------------|
| <code>file.[fresc]</code>      | input file                               |
| <code>file.o</code>            | object file                              |
| <code>a.out</code>             | linked output                            |
| <code>./fort[pid].?</code>     | temporary                                |
| <code>/usr/lib/f77pass1</code> | compiler                                 |
| <code>/lib/c1</code>           | pass 2                                   |
| <code>/lib/c2</code>           | optional optimizer                       |
| <code>/usr/lib/libF77.a</code> | intrinsic function library               |
| <code>/usr/lib/libI77.a</code> | Fortran I/O library                      |
| <code>/lib/libc.a</code>       | C library; see Section 3 of this Manual. |

## SEE ALSO

*A Portable Fortran 77 Compiler* by S. I. Feldman and P. J. Weinberger.  
`asa(1)`, `cc(1)`, `efl(1)`, `fsplit(1)`, `ld(1)`, `m4(1)`, `prof(1)`, `ratfor(1)`, `sdb(1)`.

## DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the link editor `ld(1)`.

**NAME**

factor — factor a number

**SYNOPSIS**

factor [ number ]

**DESCRIPTION**

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than  $2^{56}$  (about  $7.2 \times 10^{16}$ ) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to  $\sqrt{n}$  and occurs when  $n$  is prime or the square of a prime. It takes 1 minute to factor a prime near  $10^{14}$  on a PDP-11.

**DIAGNOSTICS**

“Ouch” for input out of range or for garbage input.

## NAME

*fget*, *fget.demon* — retrieve files from the HONEYWELL 6000

## SYNOPSIS

```
fget [ options ] [ files ]
/usr/lib/fget.demon time
```

## DESCRIPTION

*Fget* arranges to have one or more GCOS files sent to UNIX. GCOS identification must appear in the UNIX password file (see *passwd*(4)), or be supplied by the *-i* option. Normally, the files retrieved will appear in the UNIX user's current directory under the GCOS file name. *Fget.demon* is the daemon that does the actual retrieval.

The GCOS catalog from which the files are obtained depends on the form of the file name argument. If the file name has only embedded slashes, then it is assumed to be a full GCOS path name and that file is retrieved. If the file name has no embedded slashes or begins with a slash, then the GCOS catalog from which the file is retrieved is the same as the UNIX login name of the person who issues the command. If, however, a user has a different name in the third field of the GCOS "ident card image" (which image is extracted from the UNIX password file—see *passwd*(4)), this name is taken as the GCOS catalog name. Whatever GCOS catalog is finally used, the files must either have general read permission or the user must have arranged that the user ID *network* has read permission on that catalog (see *fsend*(1C)). This can be accomplished with the GCOS command:

```
filsys mc <user ID>,(r)/network/
```

The UNIX file into which the retrieved GCOS file will ultimately be written is initialized with one line containing the complete GCOS file name. If the file contains the initial line for an extended period, it means that GCOS is down or something has gone horribly wrong and you should try again.

The following options, each as a separate argument may appear in any order but must precede all file arguments.

- a* Retrieve files as ASCII (default).
- b* Retrieve files as binary.
- dir* Use *dir* as the UNIX directory into which retrieved files are written.
- file* Use *file* as the UNIX filename for the retrieved file.
- ijob,bin*  
Supply the GCOS "ident card" image as the parameter *-ijob,bin* where *job* is the GCOS job number and *bin* the GCOS bin number or any comment to the GCOS operators.
- m* When the request has been forwarded to GCOS, report by *mail*(1) the so-called *snumb* of the receiving job; mail is sent by the UNIX *dpd*(1C) daemon; there is no guarantee that the GCOS job ran or that UNIX retrieved the output. This is the default option.
- n* Do not report the forwarding of the request by *mail*(1).
- o* Print the on-line GCOS accounting output.
- t* Toss out the on-line GCOS accounting output. This is the default option.
- sn* Submit job to GCOS with service grade *n* (*n*=1, 2, 3, 4). Default is *-s1*.
- userid*  
Use *userid* as the GCOS catalog name for all files.

The GCOS job to send the requested files to UNIX is sent by the *dpd*(1C) daemon. Receiving these files is then done by a corresponding retrieval daemon, *fget.demon*, which stays alive for a minimum of *time* seconds,

(default 360), or until it has successfully retrieved one or more files. The file `glock` in the spooling directory `/usr/spool/dpd` is used to prevent two daemons from becoming active simultaneously. After the program has successfully set the lock, it forks and the main path exits, thus spawning the daemon. GRTS is interrogated for any output for the daemon's station-id. If none, `fget.demon` will wait up to `time` seconds, interrogating GRTS every minute or so to see if any output has arrived. All problems and successful transactions are recorded in the `errors` file in the spooling directory.

To restart `fget.demon` (in the case of hardware or software malfunction), it is necessary to first kill the old `fget.demon` (if still alive), and remove the lock file (if present), before initiating `fget.demon`. This should be done automatically by `/etc/rc` when the system is brought up, in case there are any files waiting to come over.

#### EXAMPLES

The command:

```
fget -u gcosme -t -n -d /usr/me/test file1 file2
```

will retrieve the GCOS files `gcosme/file1` and `gcosme/file2`, as the UNIX files `/usr/me/test/file1` and `/usr/me/test/file2`, respectively, but will not generate any mail or GCOS accounting output as a result of the transaction.

#### FILES

|                               |                                            |
|-------------------------------|--------------------------------------------|
| <code>/etc/passwd</code>      | user's identification and GCOS ident card. |
| <code>/usr/lib/dpd</code>     | sending daemon.                            |
| <code>/usr/spool/dpd/*</code> | spool area.                                |
| <code>/dev/dn?</code>         | ACU device.                                |
| <code>/dev/du?</code>         | DATA-PHONE set.                            |
| <code>/dev/vpb?</code>        | Bottom VPM device to interface to KMC11-B. |
| <code>/dev/vpm?</code>        | Top VPM device to interface to KMC11-B.    |

#### SEE ALSO

`dpd(1C)`, `dpr(1C)`, `fsend(1C)`, `passwd(4)`.

**NAME**

*file* — determine file type

**SYNOPSIS**

*file* [ **-c** ] [ **-f** *ffile* ] [ **-m** *mfile* ] *arg* ...

**DESCRIPTION**

*File* performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable *a.out*, *file* will print the version stamp, provided it is greater than 0 (see *ld(1)*).

If the **-f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

*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. Commentary at the beginning of */etc/magic* explains its format.

The **-m** option instructs *file* to use an alternate magic file.

The **-c** flag causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **-c**.

## NAME

find — find files

## SYNOPSIS

find path-name-list expression

## DESCRIPTION

*Find* recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) seeking files that match a boolean *expression* written in the primaries given below. 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*.

- name *file* True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for [, ? and \*).
- perm *onum* True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared:  
(flags&onum) == onum
- type *c* True if the type of the file is *c*, where *c* is *b*, *c*, *d*, *p*, or *f* for block special file, character special file, directory, fifo (a.k.a named pipe), or plain file.
- 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* file, it is taken as a user ID.
- group *gname* True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the */etc/group* file, it is taken as a group ID.
- size *n* True if the file is *n* blocks long (512 bytes per block).
- atime *n* True if the file has been accessed in *n* days.
- mtime *n* True if the file has been modified in *n* days.
- ctime *n* True if the file has been changed in *n* days.
- exec *cmd* True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current path name.
- ok *cmd* Like —exec except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing y.
- print Always true; causes the current path name to be printed.
- cpio *device* Write the current file on *device* in *cpio*(4) format (5120 byte records).
- newer *file* True if the current file has been modified more recently than the argument *file*.
- ( *expression* ) True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

- 1) The negation of a primary (! is the unary *not* operator).
- 2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
- 3) Alternation of primaries (-o is the *or* operator).

**EXAMPLE**

To remove all files named **a.out** or **\*.o** that have not been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

**FILES**

/etc/passwd, /etc/group

**SEE ALSO**

cpio(1), sh(1), test(1), stat(2), cpio(4), fs(4).

## NAME

*f*send — send files to the HONEYWELL 6000

## SYNOPSIS

*f*send [ options ] [ files ]

## DESCRIPTION

*F*send arranges to have one or more UNIX files sent to HONEYWELL GCOS. GCOS identification must appear in the UNIX password file (see *passwd*(4)), or be supplied by the *-i* option. If no names appear, the standard input is sent; thus *f*send may be used as a filter.

Normally, the catalog on the HONEYWELL file system in which the new file will appear is the same as the UNIX login name of the person who issues the command. If, however, a user has a different name in the third field of the GCOS "ident card image" (which image is extracted from the UNIX password file; see *passwd*(4)), this name is taken as the GCOS catalog name. Whatever GCOS catalog is finally used, the user must have arranged that the user ID "network" has create permission on that catalog, or read and write permission on the individual files. The latter is more painful but preferred if access to other files in the catalog is to be fully controlled. This can be accomplished with the GCOS commands:

```
filsys mc <user ID>,c/network/,m/<user ID>/
```

or

```
filsys cf <file>,w/network/,b/<initial-size>,unlimited/
```

The name of the GCOS file is ordinarily the same as the name of the UNIX file. When the standard input is sent, the GCOS file is normally taken to be *pipe.end*.

The following options, each as a separate argument, may appear in any order but must precede all file name arguments.

- a* Send succeeding files as ASCII (default). If the last character of the file is not a new-line, one is added. All other characters are preserved.
- b* Send succeeding files as binary. Each UNIX byte is right justified in a GCOS byte and the bytes packed into 120-byte logical records (30 GCOS words). The last record is padded out with NULs.
- c* Make copies of the files to be sent before returning to the user.
- r* Remove the files after sending them.
- f**file* Use *file* as the GCOS file name for the file being sent.
- i**job,bin*  
Supply the GCOS "ident card" image as the parameter *-i**job,bin* where *job* is the GCOS job number and *bin* the GCOS bin number or any comment to the GCOS operators.
- m* When transmission is complete, report by *mail*(1) the so-called *snumb* of the receiving GCOS job. The mail is sent by the UNIX daemon; there is no guarantee that the GCOS job ran successfully. This is the default option.
- n* Do not report the completion of transmission by *mail*(1).
- o* Print the on-line GCOS accounting output.
- t* Toss out the on-line GCOS accounting output. This is the default option.
- sn* Submit job to GCOS with service grade *n* (*n*=1, 2, 3, 4). Default is *-s1*.
- u**userid*  
Use *userid* as the GCOS catalog name for all files.



**-x** Send succeeding files to be archived by the GCOS archive command.

**EXAMPLE**

The command:

```
fsend -t -u unixsup -b -f gfile ufile
```

will send the binary UNIX file **ufile** to become the GCOS file **unixsup/gfile**, and will not produce any on-line GCOS accounting output.

**FILES**

|                  |                                            |
|------------------|--------------------------------------------|
| /etc/passwd      | user's identification and GCOS ident card. |
| /usr/lib/dpd     | sending daemon.                            |
| /usr/spool/dpd/* | spool area.                                |

**SEE ALSO**

dpd(1C), dpr(1C), fget(1C), gcat(1C), mail(1).

**NAME**

fsplit — split f77, ratfor, or efl files

**SYNOPSIS**

fsplit options files

**DESCRIPTION**

*Fsplit* splits the named *file(s)* into separate files, with one procedure per file. A procedure includes *blockdata*, *function*, *main*, *program*, and *subroutine* program segments. Procedure *X* is put in file *X.f*, *X.r*, or *X.e* depending on the language option chosen, with the following exceptions: *main* is put in the file *MAIN.[efr]* and unnamed *blockdata* segments in the files *blockdataN.[efr]* where *N* is a unique integer value for each file.

The following *options* pertain:

- f (default) Input files are *f77*.
- r Input files are *ratfor*.
- e Input files are *Efl*.
- s Strip *f77* input lines to 72 or fewer characters with trailing blanks removed.

**SEE ALSO**

csplit(1), efl(1), f77(1), ratfor(1), split(1).

## NAME

`gcat` — send phototypesetter output to the HONEYWELL 6000

## SYNOPSIS

`gcat` [ options ] [ files ]

## DESCRIPTION

`Gcat` arranges to have `troff(1)` output sent to the phototypesetter or debugging devices (STARE or line printer) attached to the HONEYWELL system. GCOS identification must appear in the UNIX password file (see `passwd(4)`), or be supplied by the `-i` option. If no file name appears, the standard input is sent; thus `gcat` may be used as an output pipe for `troff(1)`.

The option `-g` (for GCOS) must be used with the `troff(1)` command to make things work properly. This command string sends output to the GCOS phototypesetter:

```
troff -g file | gcat
```

The following options, each as a separate argument, and in any combination (multiple outputs are permitted), may be given after `gcat`:

- `-ph` Send output to the phototypesetter. This is a default option.
- `-st` Send output to STARE for fast turn-around.
- `-tx` Send output as text to the line printer (useful for checking spelling, hyphenation, pagination, etc.).
- `-du` Send output to the line printer, dummied up to make the format correct. Because many characters are dropped, the output is unreadable, but useful for seeing the shape (margins, etc.) of the document.
- `-c` Make a copy of the file to be sent before returning to the user.
- `-r` Remove the file after sending it.
- `-ffile` Use `file` as a dummy file name to report back in the mail. (This is useful for distinguishing multiple runs, especially when `gcat` is being used as a filter).
- `-ijob,bin`  
Supply the GCOS “ident card” image as the parameter `-ijob,bin` where `job` is the GCOS job number and `bin` the GCOS bin number or any comment to the GCOS operators.
- `-m` When transmission is complete, report by `mail(1)` the so-called `snumb` of the receiving GCOS job. The mail is sent by the UNIX daemon; there is no guarantee that the GCOS job ran successfully. This is a default option.
- `-n` Do not report the completion of transmission by `mail(1)`.
- `-o` Print the on-line GCOS accounting output.
- `-t` Toss out the on-line GCOS accounting output. This is a default option.
- `-sn` Submit job to GCOS with service grade `n` (`n=1, 2, 3, 4`). Default is `-s1`.

If none of the output options are specified, phototypesetter output (`-ph`) is assumed by default.

## EXAMPLE

The command:

```
troff -g myfile | gcat -st -im1234,m567,myname -fmyfile
```

will send the output of `troff(1)` to STARE, with the GCOS “ident card” specifying “M1234,M567,MYNAME”, and will report back that `myfile` has been sent.

**FILES**

|                  |                                            |
|------------------|--------------------------------------------|
| /etc/passwd      | user's identification and GCOS ident card. |
| /usr/lib/dpd     | sending daemon.                            |
| /usr/spool/dpd/* | spool area.                                |

**SEE ALSO**

dpd(1C), dpr(1C), fget(1C), fsend(1C), troff(1).

**NAME**

gcosmail — send mail to HIS user

**SYNOPSIS**

gcosmail [ option ... ] [ HISuserid ... ]

**DESCRIPTION**

*Gcosmail* takes the standard input up to an end of file and sends it as mail to the named users on the HONEYWELL 6000 system, using the HIS *mail* command. The following options are recognized by *gcosmail*:

- file* Use *file* as a dummy file name to report back in the mail. (This is useful for distinguishing multiple runs).
- ijob,bin*  
Supply the GCOS "ident card" image as the parameter *-ijob,bin* where *job* is the GCOS job number and *bin* the GCOS bin number or any comment to the GCOS operators.
- m* When transmission is complete, report by *mail(1)* the so-called *smumb* of the receiving GCOS job. The mail is sent by the UNIX daemon; there is no guarantee that the GCOS job ran successfully. This is a default option.
- n* Do not report the completion of transmission by *mail(1)*.
- o* Print the on-line GCOS accounting output.
- t* Toss out the on-line GCOS accounting output. This is a default option.
- sn* Submit job to GCOS with service grade *n* (*n*=1, 2, 3, 4). Default is *-s1*.

**FILES**

|                  |                                            |
|------------------|--------------------------------------------|
| /etc/passwd      | user's identification and GCOS ident card. |
| /usr/lib/dpd     | sending daemon.                            |
| /usr/spool/dpd/* | spool area.                                |

**SEE ALSO**

dpd(1C), dpr(1C), fsend(1C).

## NAME

hpd, erase, hardcopy, tekset, td — graphical device routines and filters

## SYNOPSIS

**hpd** [-options] [GPS file ...]  
**erase**  
**hardcopy**  
**tekset**  
**td** [-eurn] [GPS file ...]

## DESCRIPTION

All of the commands described below reside in `/usr/bin/graf` (see *graphics(1G)*).

- hpd** Hpd translates a GPS (see *gps(4)*), to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in *file* unless the `-u` or `-r` option is provided. If no *file* is given, the standard input is assumed. Options are:
- cn** Select character set *n*, *n* between 0 and 5 (see the *HP7221A Plotter Operating and Programming Manual, Appendix A*).
  - pn** Select pen numbered *n*, *n* between 1 and 4 inclusive.
  - rn** Window on GPS region *n*, *n* between 1 and 25 inclusive.
  - sn** Slant characters *n* degrees clockwise from the vertical.
  - u** Window on the entire GPS universe.
  - xdn** Set x displacement of the viewport's lower left corner to *n* inches.
  - xvn** Set width of viewport to *n* inches.
  - yd** Set y displacement of the viewport's lower left corner to *n* inches.
  - yvn** Set height of viewport to *n* inches.
- erase** *Erase* sends characters to a Tektronix 4010 series storage terminal to erase the screen.
- hardcopy** When issued at a Tektronix display terminal with a hard copy unit, *hardcopy* generates a screen copy on the unit.
- tekset** *Tekset* sends characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.
- td** *Td* translates a GPS to scope code for a Tektronix 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in *file* unless the `-u` or `-r` option is provided. If no *file* is given, the standard input is assumed. Options are:
- e** Do not erase screen before initiating display.
  - rn** Display GPS region *n*, *n* between 1 and 25 inclusive.
  - u** Display the entire GPS universe.

## SEE ALSO

*ged(1G)*, *graphics(1G)*, *gps(4)*.

## NAME

`ged` — graphical editor

## SYNOPSIS

`ged [-euRrn] [GPS file ...]`

## DESCRIPTION

*Ged* is an interactive graphical editor used to display, construct, and edit GPS files on Tektronix 4010 series display terminals. If GPS *file(s)* are given, *ged* reads them into an internal display buffer and displays the buffer. The GPS in the buffer can then be edited. If `-` is given as a file name, *ged* reads a GPS from the standard input.

*Ged* accepts the following command line options:

- `e` Do not erase the screen before the initial display.
- `rn` Display region number *n*.
- `u` Display the entire GPS *universe*.
- `R` Restricted shell invoked on use of `!`.

A GPS file is composed of instances of three graphical objects: *lines*, *arc*, and *text*. *Arc* and *lines* objects have a start point, or *object-handle*, followed by zero or more points, or *point-handles*. *Text* has only an object-handle. The objects are positioned within a Cartesian plane, or *universe*, having 64K ( $-32K$  to  $+32K$ ) points, or *universe-units*, on each axis. The universe is divided into 25 equal sized areas called *regions*. Regions are arranged in five rows of five squares each, numbered 1 to 25 from the lower left of the universe to the upper right.

*Ged* maps rectangular areas, called *windows*, from the universe onto the display screen. Windows allow the user to view pictures from different locations and at different magnifications. The *universe-window* is the window with minimum magnification, i.e. the window that views the entire universe. The *home-window* is the window that completely displays the contents of the display buffer.

## COMMANDS

*Ged* commands are entered in *stages*. Typically each stage ends with a `<cr>` (return). Prior to the final `<cr>` the command may be aborted by typing *rubout*. The input of a stage may be edited during the stage using the erase and kill characters of the calling shell. The prompt `*` indicates that *ged* is waiting at stage 1.

Each command consists of a subset of the following stages:

1. *Command line*  
A *command line* consists of a *command name* followed by *argument(s)* followed by a `<cr>`. A *command name* is a single character. *Command arguments* are either *option(s)* or a *file-name*. *Options* are indicated by a leading `-`.
2. *Text*  
*Text* is a sequence of characters terminated by an unescaped `<cr>`. (120 lines of text maximum.)
3. *Points*  
*Points* is a sequence of one or more screen locations (maximum of 30) indicated either by the terminal crosshairs or by name. The prompt for entering *points* is the appearance of the crosshairs. When the crosshairs are visible, typing:  
`sp` (space) enters the current location as a *point*. The *point* is identified with a number.

- \$n** enters the previous *point* numbered *n*.
- >x** labels the last *point* entered with the upper case letter *x*.
- \$x** enters the *point* labeled *x*.
- .** establishes the previous *points* as the current *points*. At the start of a command the previous *points* are those locations given with the previous command.
- =** echoes the current *points*.
- \$.n** enters the *point* numbered *n* from the previous *points*.
- #** erases the last *point* entered.
- @** erases all of the *points* entered.
4. *Pivot* The *pivot* is a single location, entered by typing <cr> or by using the \$ operator, and indicated with a \*.
5. *Destination* The *destination* is a single location entered by typing <cr> or by using \$.

### COMMAND SUMMARY

In the summary, characters typed by the user are printed in **bold**. Command stages are printed in *italics*. Arguments surrounded by brackets “[ ]” are optional. Parentheses “( )” surrounding arguments separated by “or” means that exactly one of the arguments must be given.

#### Construct commands:

|                 |                                                                                    |
|-----------------|------------------------------------------------------------------------------------|
| <b>Arc</b>      | [ <b>-echo,style,weight</b> ] <i>points</i>                                        |
| <b>Box</b>      | [ <b>-echo,style,weight</b> ] <i>points</i>                                        |
| <b>Circle</b>   | [ <b>-echo,style,weight</b> ] <i>points</i>                                        |
| <b>Hardware</b> | [ <b>-echo</b> ] <i>text points</i>                                                |
| <b>Lines</b>    | [ <b>-echo,style,weight</b> ] <i>points</i>                                        |
| <b>Text</b>     | [ <b>-angle,echo,height,mid-point,right-point,text,weight</b> ] <i>text points</i> |

#### Edit commands:

|               |                                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| <b>Delete</b> | ( <b>- (universe or view) or points</b> )                                            |
| <b>Edit</b>   | [ <b>-angle,echo,height,style,weight</b> ] ( <b>- (universe or view) or points</b> ) |
| <b>Kopy</b>   | [ <b>-echo,points,x</b> ] <i>points pivot destination</i>                            |
| <b>Move</b>   | [ <b>-echo,points,x</b> ] <i>points pivot destination</i>                            |
| <b>Rotate</b> | [ <b>-angle,echo,kopy,x</b> ] <i>points pivot destination</i>                        |
| <b>Scale</b>  | [ <b>-echo,factor,kopy,x</b> ] <i>points pivot destination</i>                       |

#### View commands:

|                       |                                           |
|-----------------------|-------------------------------------------|
| <b>coordinates</b>    | <i>points</i>                             |
| <b>erase</b>          |                                           |
| <b>new-display</b>    |                                           |
| <b>object-handles</b> | ( <b>- (universe or view) or points</b> ) |



**point-handles** ( - (labelled-points or universe or view) or *points* )  
**view** ( - (home or universe or region) or [-x] *pivot destination* )  
**x** [-view] *points*  
**zoom** [-out] *points*

**Other commands:**

quit or Quit

**read** [-angle,echo,height,mid-point,right-point,text,weight]  
*file-name* [*destination*]

**set** [-angle,echo,factor,height,kopy,mid-point,points,  
right-point,style,text,weight,x]

**write** *file-name*

!command

?

**Options:**

*Options* specify parameters used to construct, edit, and view graphical objects. If a parameter used by a command is not specified as an *option*, the default value for the parameter will be used (see set below). The format of command *options* is

-option [,option]

where *option* is *keyletter*[*value*]. Flags take on the *values* of true or false indicated by + and - respectively. If no *value* is given with a flag, true is assumed.

**Object options:**

**anglen** Angle of *n* degrees.  
**echo** When true, echo additions to the display buffer.  
**factorn** Scale factor is *n* percent.  
**heightn** Height of *text* is *n* universe-units ( $0 \leq n < 1280$ ).  
**kopy** When true, copy rather than move.  
**mid-point** When true, mid-point is used to locate text string.  
**points** When true, operate on points otherwise operate on objects.  
**right-point** When true, right-point is used to locate *text* string.  
**styletype** Line style set to one of following *types*:

|           |             |
|-----------|-------------|
| <b>so</b> | solid       |
| <b>da</b> | dashed      |
| <b>dd</b> | dot-dashed  |
| <b>do</b> | dotted      |
| <b>ld</b> | long-dashed |

|                   |                                                                 |
|-------------------|-----------------------------------------------------------------|
| <b>text</b>       | When false, <i>text</i> strings are outlined rather than drawn. |
| <b>weighttype</b> | Sets line weight to one of following <i>types</i> :             |
|                   | <b>n</b> narrow                                                 |
|                   | <b>m</b> medium                                                 |
|                   | <b>b</b> bold                                                   |

**Area options:**

|                 |                                             |
|-----------------|---------------------------------------------|
| <b>home</b>     | Reference the home-window.                  |
| <b>out</b>      | Reduce magnification.                       |
| <b>regionn</b>  | Reference region <i>n</i> .                 |
| <b>universe</b> | Reference the universe-window.              |
| <b>view</b>     | Reference those objects currently in view.  |
| <b>x</b>        | Indicate the center of the referenced area. |

**COMMAND DESCRIPTIONS****Construct commands:****Arc and Lines**

behave similarly. Each consists of a *command line* followed by *points*. The first *point* entered is the object-handle. Successive *points* are point-handles. Lines connects the handles in numerical order. Arc fits a curve to the handles (currently a maximum of 3 points will be fit with a circular arc; splines will be added in a later version).

**Box and Circle**

are special cases of Lines and Arc, respectively. Box generates a rectangle with sides parallel to the universe axes. A diagonal of the rectangle would connect the first *point* entered with the last *point*. The first *point* is the object-handle. Point-handles are created at each of the vertices. Circle generates a circular arc centered about the *point* numbered zero and passing through the last *point*. The circle's object-handle coincides with the last *point*. A point-handle is generated 180 degrees around the circle from the object-handle.

**Text and Hardware**

generate *text* objects. Each consists of a *command line*, *text* and *points*. *Text* is a sequence of characters delimited by <cr>. Multiple lines of text may be entered by preceding a cr with a backslash (i.e. \cr). The Text command creates software generated characters. Each line of software text is treated as a separate *text* object. The first *point* entered is the object-handle for the first line of text. The Hardware command sends the characters in *text* uninterpreted to the terminal.

**Edit commands:**

Edit commands operate on portions of the display buffer called *defined-areas*. A defined-area is referenced either with an area *option* or interactively. If an area *option* is not given, the perimeter of the defined-area is indicated by *points*. If no *point* is entered, a small defined-area is built around the location of the <cr>. This is useful to reference a single *point*. If only one *point* is entered, the location of the <cr> is taken in conjunction with the *point* to indicate a diagonal of a rectangle. A defined-area referenced by *points* will be outlined with dotted lines.

**Delete**

removes all objects whose object-handle lies within a defined-area. The universe option removes all objects and erases the screen.

**Edit** modifies the parameters of the objects within a defined-area. Parameters that can be edited are:

angle angle of *text*  
 height height of *text*  
 style style of *lines* and *arc*  
 weight weight of *lines*, *arc*, and *text*.

**Kopy (or Move)**

copies (or moves) object- and/or point-handles within a defined-area by the displacement from the *pivot* to the *destination*.

**Rotate**

rotates objects within a defined-area around the *pivot*. If the *kopy* flag is true then the objects are copied rather than moved.

**Scale**

For objects whose object-handles are within a defined-area, point displacements from the *pivot* are scaled by factor percent. If the *kopy* flag is true then the objects are copied rather than moved.

### View commands:

**coordinates**

prints the location of *point(s)* in universe- and screen-units.

**erase**

clears the screen (but not the display buffer).

**new-display**

erases the screen then displays the display buffer.

**object-handles (or point-handles)**

labels object- (and/or point-handles) that lie within the defined-area with **O** (or **P**). *point-handles* identifies labelled points when the *labelled-points* flag is true.

**view** moves the window so that the universe point corresponding to the *pivot* coincides with the screen point corresponding to the *destination*. Options for *home*, *universe*, and *region* display particular windows in the universe.

**x** indicates the center of a defined-area. Option *view* indicates the center of the screen.

**zoom**

decreases (zoom out) or increases the magnification of the viewing window based on the defined-area. For increased magnification, the window is set to circumscribe the defined-area. For a decrease in magnification the current window is inscribed within the defined-area.

### Other commands:

**quit or Quit**

exit from *ged*. *quit* responds with ? if the display buffer has not been written since the last modification.

**read** inputs the contents of a file. If the file contains a GPS it is read directly. If the file contains text it is converted into *text* object(s). The first line of a text file begins at *destination*.

**set** when given *option(s)* resets default parameters, otherwise it prints current default values.

**write**

outputs the contents of the display buffer to a file.

! escapes *ged* to execute a UNIX command.

? lists *ged* commands.

**SEE ALSO**

*gdev(1G)*, *graphics(1G)*, *sh(1)*, *gps(4)*.

*An Introduction to the Graphical Editor* in the *UNIX System Graphics Guide*.

## NAME

get — get a version of an SCCS file

## SYNOPSIS

```
get [-rSID] [-ccutoff] [-ilist] [-xlist] [-aseq-no.] [-k] [-e]
[-l[p]] [-p] [-m] [-n] [-s] [-b] [-g] [-t] file ...
```

## DESCRIPTION

*Get* generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with `-`. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with `s.`) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading `s.`; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

`-rSID` The *SCCS IDentification* string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the `-e` keyletter is also used), as a function of the SID specified.

`-ccutoff` *Cutoff* date-time, in the form:

```
YY[MM[DD[HH[MM[SS]]]]]
```

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-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 pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: `"-c77/2/2 9:22:25"`. Note that this implies that one may use the `%E%` and `%U%` identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

```
~!get "-c%E% %U%" s.file
```

`-e` Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The `-e` keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *gets* for editing on the same SID until *delta* is executed or the `j` (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of *get* `-e` for different SIDs is always allowed.

If the *g-file* generated by *get* with an `-e` keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the `-k` keyletter in place of the `-e` keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin(1)*) are enforced when the `-e` keyletter is used.

- `-b` Used with the `-e` keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the `b` flag is not present in the file (see *admin(1)*) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)  
Note: A branch *delta* may always be created from a non-leaf *delta*.
- `-ilist` A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:  

```
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID - SID
```

 SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.
- `-xlist` A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the `-i` keyletter for the *list* format.
- `-k` Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The `-k` keyletter is implied by the `-e` keyletter.
- `-l[p]` Causes a delta summary to be written into an *l-file*. If `-lp` is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.
- `-p` Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the `-s` keyletter is used, in which case it disappears.
- `-s` Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.
- `-m` Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.
- `-n` Causes each generated text line to be preceded with the `%M%` identification keyword value (see below). The format is: `%M%` value, followed by a horizontal tab, followed by the text line. When both the `-m` and `-n` keyletters are used, the format is: `%M%` value, followed by a horizontal tab, followed by the `-m` keyletter generated format.
- `-g` Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.
- `-t` Used to access the most recently created ("top") delta in a given release (e.g., `-r1`), or release and level (e.g., `-r1.2`).

—*aseq-no.* The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile(5)*). This keyletter is used by the *comb(1)* command; it is not a generally useful keyletter, and users should not use it. If both the *-r* and *-a* keyletters are specified, the *-a* keyletter is used. Care should be taken when using the *-a* keyletter in conjunction with the *-e* keyletter, as the SID of the delta to be created may not be what one expects. The *-r* keyletter can be used with the *-a* and *-e* keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the *-e* keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the *-i* keyletter is used included deltas are listed following the notation "Included"; if the *-x* keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

| SID*<br>Specified | —b Keyletter<br>Used† | Other<br>Conditions                            | SID<br>Retrieved | SID of Delta<br>to be Created |
|-------------------|-----------------------|------------------------------------------------|------------------|-------------------------------|
| none‡             | no                    | R defaults to mR                               | mR.mL            | mR.(mL+1)                     |
| none‡             | yes                   | R defaults to mR                               | mR.mL            | mR.mL.(mB+1).1                |
| R                 | no                    | R > mR                                         | mR.mL            | R.1***                        |
| R                 | no                    | R = mR                                         | mR.mL            | mR.(mL+1)                     |
| R                 | yes                   | R > mR                                         | mR.mL            | mR.mL.(mB+1).1                |
| R                 | yes                   | R = mR                                         | mR.mL            | mR.mL.(mB+1).1                |
| R                 | —                     | R < mR and<br>R does <i>not</i> exist          | hR.mL**          | hR.mL.(mB+1).1                |
| R                 | —                     | Trunk succ.#<br>in release > R<br>and R exists | R.mL             | R.mL.(mB+1).1                 |
| R.L               | no                    | No trunk succ.                                 | R.L              | R.(L+1)                       |
| R.L               | yes                   | No trunk succ.                                 | R.L              | R.L.(mB+1).1                  |
| R.L               | —                     | Trunk succ.<br>in release ≥ R                  | R.L              | R.L.(mB+1).1                  |
| R.L.B             | no                    | No branch succ.                                | R.L.B.mS         | R.L.B.(mS+1)                  |
| R.L.B             | yes                   | No branch succ.                                | R.L.B.mS         | R.L.(mB+1).1                  |
| R.L.B.S           | no                    | No branch succ.                                | R.L.B.S          | R.L.B.(S+1)                   |
| R.L.B.S           | yes                   | No branch succ.                                | R.L.B.S          | R.L.(mB+1).1                  |
| R.L.B.S           | —                     | Branch succ.                                   | R.L.B.S          | R.L.(mB+1).1                  |

\* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

\*\* "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

- \*\*\* This is used to force creation of the *first* delta in a *new* release.
- # Successor.
- † The *-b* keyletter is effective only if the *b* flag (see *admin(1)*) is present in the file. An entry of *-* means "irrelevant".
- ‡ This case applies if the *d* (default SID) flag is *not* present in the file. If the *d* flag is present in the file, then the SID obtained from the *d* flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

### IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

| <i>Keyword</i> | <i>Value</i>                                                                                                                                                                                                                     |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| %M%            | Module name: either the value of the <i>m</i> flag in the file (see <i>admin(1)</i> ), or if absent, the name of the SCCS file with the leading <i>s.</i> removed.                                                               |
| %I%            | SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.                                                                                                                                                               |
| %R%            | Release.                                                                                                                                                                                                                         |
| %L%            | Level.                                                                                                                                                                                                                           |
| %B%            | Branch.                                                                                                                                                                                                                          |
| %S%            | Sequence.                                                                                                                                                                                                                        |
| %D%            | Current date (YY/MM/DD).                                                                                                                                                                                                         |
| %H%            | Current date (MM/DD/YY).                                                                                                                                                                                                         |
| %T%            | Current time (HH:MM:SS).                                                                                                                                                                                                         |
| %E%            | Date newest applied delta was created (YY/MM/DD).                                                                                                                                                                                |
| %G%            | Date newest applied delta was created (MM/DD/YY).                                                                                                                                                                                |
| %U%            | Time newest applied delta was created (HH:MM:SS).                                                                                                                                                                                |
| %Y%            | Module type: value of the <i>t</i> flag in the SCCS file (see <i>admin(1)</i> ).                                                                                                                                                 |
| %F%            | SCCS file name.                                                                                                                                                                                                                  |
| %P%            | Fully qualified SCCS file name.                                                                                                                                                                                                  |
| %Q%            | The value of the <i>q</i> flag in the file (see <i>admin(1)</i> ).                                                                                                                                                               |
| %C%            | Current line number. This keyword is intended for identifying messages output by the program such as "this shouldn't have happened" type errors. It is <i>not</i> intended to be used on every line to provide sequence numbers. |
| %Z%            | The 4-character string <i>@(#)</i> recognizable by <i>what(1)</i> .                                                                                                                                                              |
| %W%            | A shorthand notation for constructing <i>what(1)</i> strings for UNIX program files. %W% = %Z%%M%<horizontal-tab>%I%                                                                                                             |
| %A%            | Another shorthand notation for constructing <i>what(1)</i> strings for non-UNIX program files. %A% = %Z%%Y% %M% %I%%Z%                                                                                                           |

### FILES

Several auxiliary files may be created by *get*. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form *s.module-name*, the auxiliary files are named by replacing the leading *s* with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the *s.* prefix. For example, *s.xyz.c*, the auxiliary file names would be *xyz.c*, *l.xyz.c*, *p.xyz.c*, and *z.xyz.c*, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the *-p* keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the *-k* keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the



current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the `-l` keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

- a. A blank character if the delta was applied;  
\* otherwise.
- b. A blank character if the delta was applied or wasn't applied and ignored;  
\* if the delta wasn't applied and wasn't ignored.
- c. A code indicating a "special" reason why the delta was or was not applied:  
"I": Included.  
"X": Excluded.  
"C": Cut off (by a `-c` keyletter).
- d. Blank.
- e. SCCS identification (SID).
- f. Tab character.
- g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
- h. Blank.
- i. Login name of person who created *delta*.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an `-e` keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an `-e` keyletter for the same SID until *delta* is executed or the joint edit flag, `j`, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the `-i` keyletter argument if it was present, followed by a blank and the `-x` keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

#### SEE ALSO

*admin*(1), *delta*(1), *help*(1), *prs*(1), *what*(1), *sccsfile*(4).  
*Source Code Control System* in the *UNIX System Support Tools Guide*.

#### DIAGNOSTICS

Use *help*(1) for explanations.

#### BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, then only one file may be named when the `-e` keyletter is used.

**NAME**

getopt -- parse command options

**SYNOPSIS**

```
set -- `getopt optstring $*`
```

**DESCRIPTION**

*Getopt* is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *Optstring* is a string of recognized option letters (see *getopt(3C)*); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option -- is used to delimit the end of the options. If it is used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it; in either case, *getopt* will place it at the end of the options. The shell's positional parameters (\$1 \$2 ...) are reset so that each option is preceded by a - and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

**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:

```
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 as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

**SEE ALSO**

sh(1), getopt(3C).

**DIAGNOSTICS**

*Getopt* prints an error message on the standard error when it encounters an option letter not included in *optstring*.

## NAME

graph — draw a graph

## SYNOPSIS

graph [ options ]

## DESCRIPTION

*Graph* with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *tplot(1G)* filters.

If the coordinates of a point are followed by a non-numeric 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 new-lines.

The following options are recognized, each as a separate argument:

- `-a` Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by `-x`).
- `-b` Break (disconnect) the graph after each label in the input.
- `-c` Character string given by next argument is default label for each point.
- `-g` Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).
- `-l` Next argument is label for graph.
- `-m` Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (e.g., the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).
- `-s` Save screen, don't erase before plotting.
- `-x [ 1 ]` If 1 is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x limits. Third argument, if present, is grid spacing on x axis. Normally these quantities are determined automatically.
- `-y [ 1 ]` Similarly for y.
- `-h` Next argument is fraction of space for height.
- `-w` Similarly for width.
- `-r` Next argument is fraction of space to move right before plotting.
- `-u` Similarly to move up before plotting.
- `-t` Transpose horizontal and vertical axes. (Option `-x` now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the `-s` option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

## SEE ALSO

graphics(1G), spline(1G), tplot(1G).

## BUGS

*Graph* stores all points internally and drops those for which there isn't room.

Segments that run out of bounds are dropped, not windowed.

Logarithmic axes may not be reversed.

**NAME**

`graphics` — access graphical and numerical commands

**SYNOPSIS**

`graphics [ -r ]`

**DESCRIPTION**

*Graphics* appends the path name `/usr/bin/graf` to the current `$PATH` value, changes the primary shell prompt to `%`, and executes a new shell. The directory `/usr/bin/graf` contains all of the Graphics subsystem commands. If the `-r` option is given, access to the graphical commands is created in a restricted environment; that is, `$PATH` is set to `:/rbin:/usr/rbin:/bin:/usr/bin:/usr/bin/graf` and the restricted shell, `rsh`, is invoked. To restore the environment that existed prior to issuing the *graphics* command, type EOT (control-d on most terminals). To logoff from the *graphics* environment, type `quit`.

The command line format for a command in *graphics* is *command name* followed by *argument(s)*. An *argument* may be a *file name* or an *option string*. A *file name* is the name of any UNIX file except those beginning with `-`. The *file name* `-` is the name for the standard input. An *option string* consists of `-` followed by one or more *option(s)*. An *option* consists of a keyletter possibly followed by a value. *Options* may be separated by commas.

The graphical commands have been partitioned into four groups.

Commands that manipulate and plot numerical data; see *stat(1G)*.

Commands that generate tables of contents; see *toc(1G)*.

Commands that interact with graphical devices; see *gdev(1G)* and *ged(1G)*.

A collection of graphical utility commands; see *gutil(1G)*.

A list of the *graphics* commands can be generated by typing `whatis` in the *graphics* environment.

**SEE ALSO**

*gdev(1G)*, *ged(1G)*, *gutil(1G)*, *stat(1G)*, *toc(1G)*, *gps(4)*.

*UNIX System Graphics Guide*.



**NAME**

greek — select terminal filter

**SYNOPSIS**

greek [ -Tterminal ]

**DESCRIPTION**

*Greek* is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character TELETYPE® Model 37 terminal (which is the *nroff* default terminal) for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, *greek* attempts to use the environment variable **STERM** (see *environ*(5)). The following *terminals* are recognized currently:

|         |                                           |
|---------|-------------------------------------------|
| 300     | DASI 300.                                 |
| 300-12  | DASI 300 in 12-pitch.                     |
| 300s    | DASI 300s.                                |
| 300s-12 | DASI 300s in 12-pitch.                    |
| 450     | DASI 450.                                 |
| 450-12  | DASI 450 in 12-pitch.                     |
| 1620    | Diablo 1620 (alias DASI 450).             |
| 1620-12 | Diablo 1620 (alias DASI 450) in 12-pitch. |
| 2621    | Hewlett-Packard 2621, 2640, and 2645.     |
| 2640    | Hewlett-Packard 2621, 2640, and 2645.     |
| 2645    | Hewlett-Packard 2621, 2640, and 2645.     |
| 4014    | Tektronix 4014.                           |
| hp      | Hewlett-Packard 2621, 2640, and 2645.     |
| tek     | Tektronix 4014.                           |

**FILES**

/usr/bin/300  
 /usr/bin/300s  
 /usr/bin/4014  
 /usr/bin/450  
 /usr/bin/hp

**SEE ALSO**

300(1), 4014(1), 450(1), eqn(1), hp(1), mm(1), tplot(1G), nroff(1), environ(5), greek(5), term(5).

## NAME

grep, egrep, fgrep — search a file for a pattern

## SYNOPSIS

```
grep [ options ] expression [ files ]
egrep [ options ] [ expression ] [ files ]
fgrep [ options ] [ strings ] [ files ]
```

## DESCRIPTION

Commands of the *grep* family search the input *files* (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); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

- v All lines but those matching are printed.
- x (Exact) only lines matched in their entirety are printed (*fgrep* only).
- c Only a count of matching lines is printed.
- l Only the names of files with matching lines are listed (once), separated by new-lines.
- n Each line is preceded by its relative line number in the file.
- b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.
- s The error messages produced for nonexistent or unreadable files are suppressed (*grep* only).
- e *expression*  
Same as a simple *expression* argument, but useful when the *expression* begins with a — (does not work with *grep*).
- f *file*  
The regular *expression* (*egrep*) or *strings* list (*fgrep*) is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters \$, \*, [, ^, |, (, ), and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'

*Fgrep* searches for lines that contain one of the *strings* separated by new-lines.

*Egrep* accepts regular expressions as in *ed*(1), except for \ ( and \), with the addition of:

1. A regular expression followed by + matches one or more occurrences of the regular expression.
2. A regular expression followed by ? matches 0 or 1 occurrences of the regular expression.
3. Two regular expressions separated by | or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses ( ) for grouping.

The order of precedence of operators is [], then \*?+, then concatenation, then | and new-line.

## SEE ALSO

ed(1), sed(1), sh(1).

**DIAGNOSTICS**

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

**BUGS**

Ideally there should be only one *grep*, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are limited to 256 characters; longer lines are truncated.

*Egrep* does not recognize ranges, such as [a-z], in character classes.

## NAME

gutil — graphical utilities

## SYNOPSIS

command-name [options] [files]

## DESCRIPTION

Below is a list of miscellaneous device independent utility commands found in `/usr/bin/graf`. If no *files* are given, input is from the standard input. All output is to the standard output. Graphical data is stored in GPS format; see `gps(4)`.

**bel** — send bel character to terminal

**cvrtopt** [=sstring fstring lstring tstring] [args] — options converter  
*Cvrtopt* reformats *args* (usually the command line arguments of a calling shell procedure) to facilitate processing by shell procedures. An *arg* is either a file name (a string not beginning with a `-`, or a `-` by itself) or an option string (a string of options beginning with a `-`). Output is of the form:  
`-option -option . . . file name(s)`

All options appear singularly and preceding any file names. Options that take values (e.g., `-r1.1`) or are two letters long must be described through options to *cvrtopt*.

*Cvrtopt* is usually used with *set* in the following manner as the first line of a shell procedure:

```
set - `cvrtopt =[options] $@`
```

Options to *cvrtopt* are:

**sstring** *String* accepts string values.

**fstring** *String* accepts floating point numbers as values.

**lstring** *String* accepts integers as values.

**tstring** *String* is a two letter option name that takes no value.

*String* is a one or two letter option name.

**gd** [GPS files] — GPS dump  
*Gd* prints a human readable listing of GPS.

**gtop** [-rn u] [GPS files] — GPS to *plot(4)* filter  
*Gtop* transforms a GPS into *plot(4)* commands displayable by *plot* filters. GPS objects are translated if they fall within the window that circumscribes the first *file* unless an *option* is given.  
Options:

**rn** translate objects in GPS region *n*.

**u** translate all objects in the GPS universe.

**pd** [*plot(5) files*] — *plot(4)* dump  
*Pd* prints a human readable listing of *plot(4)* format graphical commands.

**ptog** [*plot(5) files*] — *plot(4)* to GPS filter  
*Ptog* transforms *plot(4)* commands into a GPS.

**quit** — terminate session

**remcom** [files] — remove comments  
*Remcom* copies its input to its output with comments removed. Comments are as defined in C (i.e., `/* comment */`).



- whatis** [-o ] [ *names* ] - brief online documentation  
*Whatis* prints a brief description of each *name* given. If no *name* is given, then the current list of description *names* is printed. **whatis** \\* prints out every description.  
Option:
- o just print command options
- yoo** *file* - pipe fitting  
*Yoo* is a piping primitive that deposits the output of a pipeline into a *file* used in the pipeline. Note that, without *yoo*, this is not usually successful as it causes a read and write on the same file simultaneously.

**SEE ALSO**

graphics(1G), gps(4).

**NAME**

help — ask for help

**SYNOPSIS**

help [args]

**DESCRIPTION**

*Help* finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

- type 1      Begins with non-numeric, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., *ge6*, for message 6 from the *get* command).
- type 2      Does not contain numerics (as a command, such as *get*)
- type 3      Is all numeric (e.g., 212)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try “help stuck”.

**FILES**

*/usr/lib/help*            directory containing files of message text.

*/usr/lib/help/helploc* file containing locations of help files not in */usr/lib/help*.

**DIAGNOSTICS**

Use *help(1)* for explanations.

## NAME

`hp` — handle special functions of HP 2640 and 2621-series terminals

## SYNOPSIS

`hp [ -e ] [ -m ]`

## DESCRIPTION

*Hp* supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most *nroff* output. A typical use is:

```
nroff -h files ... | hp
```

Regardless of the hardware options on your terminal, *hp* tries to do sensible things with underlining and reverse line-feeds. If the terminal has the “display enhancements” feature, subscripts and superscripts can be indicated in distinct ways. If it has the “mathematical-symbol” feature, Greek and other special characters can be displayed.

The flags are as follows:

- e It is assumed that your terminal has the “display enhancements” feature, and so maximal use is made of the added display modes. Overstruck characters are presented in the Underline mode. Superscripts are shown in Half-bright mode, and subscripts in Half-bright, Underlined mode. If this flag is omitted, *hp* assumes that your terminal lacks the “display enhancements” feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode, i.e., dark-on-light, rather than the usual light-on-dark.
- m Requests minimization of output by removal of new-lines. Any contiguous sequence of 3 or more new-lines is converted into a sequence of only 2 new-lines; i.e., any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, *hp* provides the same set as does *grod(1)*, except that “not” is approximated by a right arrow, and only the top half of the integral sign is shown. The display is adequate for examining output from *neqn*.

## DIAGNOSTICS

“line too long” if the representation of a line exceeds 1,024 characters. The exit codes are 0 for normal termination, 2 for all errors.

## SEE ALSO

*300(1)*, *col(1)*, *eqn(1)*, *greek(1)*, *nroff(1)*, *tbl(1)*.

## BUGS

An “overstriking sequence” is defined as a printing character followed by a backspace followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in Inverse Video; otherwise, only the first printing character is shown (again, underlined or in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse line-feeds, backspaces) can make text “disappear”; in particular, tables generated by *tbl(1)* that contain vertical lines will often be missing the lines of text that contain the “foot” of a vertical line, unless the input to *hp* is piped through *col(1)*.

Although some terminals do provide numerical superscript characters, no attempt is made to display them.

## NAME

**hpio** — HP 2645A terminal tape file archiver

## SYNOPSIS

**hpio** -o[rc] file ...

**hpio** -i[rta] [-n count]

## DESCRIPTION

*Hpio* is designed to take advantage of the tape drives on Hewlett Packard 2645A terminals. Up to 255 UNIX files can be archived onto a tape cartridge for off-line storage or for transfer to another UNIX system. The actual number of files depends on the sizes of the files. One file of about 115,000 bytes will almost fill a tape cartridge. Almost 300 1-byte files will fit on a tape, but the terminal will not be able to retrieve files after the first 255. This manual page is not intended to be a guide for using tapes on HP 2645A terminals, but tries to give enough information to be able to create and read tape archives and to position a tape for access to a desired file in an archive.

**Hpio -o** (copy out) copies the specified *file(s)*, together with path name and status information to a tape drive on your terminal (which is assumed to be positioned at the beginning of a tape or immediately after a tape mark). The left tape drive is used by default. Each *file* is written to a separate tape file and terminated with a tape mark. When *hpio* finishes, the tape is positioned following the last tape mark written.

**Hpio -i** (copy in) extracts a file(s) from a tape drive (which is assumed to be positioned at the beginning of a file that was previously written by a **hpio -o**). The default action extracts the next file from the left tape drive.

*Hpio* always leaves the tape positioned after the last file read from or written to the tape. Tapes should always be rewound before the terminal is turned off. To rewind a tape depress the green function button, then function key 5, and then select the appropriate tape drive by depressing either function key 5 for the left tape drive or function key 6 for the right. If several files have been archived onto a tape, the tape may be positioned at the beginning of a specific file by depressing the green function button, then function key 8, followed by typing the desired file number (1-255) with no RETURN, and finally function key 5 for the left tape or function key 6 for the right. The desired file number may also be specified by a signed number relative to the current file number.

The meanings of the available options are:

- r** Use the right tape drive.
- c** Include a checksum at the end of each *file*. The checksum is always checked by **hpio -i** for each file written with this option by **hpio -o**.
- n count** The number of input files to be extracted is set to *count*. If this option is not given, *count* defaults to 1. An arbitrarily large *count* may be specified to extract all files from the tape. *Hpio* will stop at the end of data mark on the tape.
- t** Print a table of contents only. No files are created. Printed information gives the file size in bytes, the file name, the file access modes, and whether or not a checksum is included for the file.
- a** Ask before creating a file. **Hpio -i** normally prints the file size and name, creates and reads in the file, and prints a status message when the file has been read in. If a checksum is included with the file, it reports whether the checksum matched its computed value. With this option, the file size and name are printed followed by a

?. Any response beginning with y or Y will cause the file to be copied in as above. Any other response will cause the file to be skipped.

## FILES

/dev/tty??

to block messages while accessing a tape

## SEE ALSO

2645A *Display Station User's Manual*, Hewlett-Packard Company, Part Number 02645-90001.

## DIAGNOSTICS

### BREAK

An interrupt signal terminated processing.

Can't create 'file'.

File system access permissions did not allow *file* to be created.

Can't get tty options on stdout.

*Hpio* was unable to get the input-output control settings associated with the terminal.

Can't open 'file'.

*File* could not be accessed to copy it to tape.

End of Tape.

No tape record was available when a read from a tape was requested. An end of data mark is the usual reason for this, but it may also occur if the wrong tape drive is being accessed and no tape is present.

'file' not a regular file.

*File* is a directory or other special file. Only regular files will be copied to tape.

Readent = *rc*, termcnt = *tc*.

*Hpio* expected to read *rc* bytes from the next block on the tape, but the block contained *tc* bytes. This is caused by having the tape improperly positioned or by a tape block being mangled by interference from other terminal I/O.

Skip to next file failed.

An attempt to skip over a tape mark failed.

Tape mark write failed.

An attempt to write a tape mark at the end of a file failed.

Write failed.

A tape write failed. This is most frequently caused by specifying the wrong tape drive, running off the end of the tape, or trying to write on a tape that is write protected.

## WARNINGS

Tape I/O operations may copy bad data if any other I/O involving the terminal occurs. Do not attempt any type ahead while *hpio* is running. *Hpio* turns off write permissions for other users while it is running, but processes started asynchronously from your terminal can still interfere. The most common indication of this problem, while a tape is being written, is the appearance of characters on the display screen that should have been copied to tape.

The keyboard, including the terminal's BREAK key, is locked during tape write operations; the BREAK key is only functional between writes.

*Hpio* must have complete control of the attributes of the terminal to communicate with the tape drives. Interaction with commands such as *cu*(1C) may interfere and prevent successful operation.

**BUGS**

Some binary files contain sequences that will confuse the terminal.

An **hpio -i** that encounters the end of data mark on the tape (e.g., scanning the entire tape with **hpio -itn 300**), leaves the tape positioned *after* the end of data mark. If a subsequent **hpio -o** is done at this point, the data will not be retrievable. The tape must be repositioned manually using the terminal's **FIND FILE -1** operation (depress the green function button, function key 8, and then function key 5 for the left tape or function key 6 for the right tape) before the **hpio -o** is started.

If an interrupt is received by **hpio** while a tape is being written, the terminal may be left with the keyboard locked. If this happens, the terminal's **RESET TERMINAL** key will unlock the keyboard.

**NAME**

hyphen — find hyphenated words

**SYNOPSIS**

**hyphen** [ files ]

**DESCRIPTION**

*Hyphen* finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, *hyphen* may be used as a filter.

**EXAMPLE**

The following will allow the proofreading of *nroff*'s hyphenation in *textfile*.  
mm textfile | hyphen

**SEE ALSO**

mm(1), troff(1).

**BUGS**

*Hyphen* can't cope with hyphenated *italic* (i.e., underlined) words; it will often miss them completely, or mangle them.

*Hyphen* occasionally gets confused, but with no ill effects other than spurious extra output.

**NAME**

**id** — print user and group IDs and names

**SYNOPSIS**

**id**

**DESCRIPTION**

*Id* writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

**SEE ALSO**

logname(1), getuid(2).



**NAME**

**ipcrm** — remove a message queue, semaphore set or shared memory id

**SYNOPSIS**

**ipcrm** [ *options* ]

**DESCRIPTION**

*Ipcrm* will remove one or more specified message, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

- q** *msgid* removes the message queue identifier *msgid* from the system and destroys the message queue and data structure associated with it.
- m** *shmid* removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure 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 structure associated with it.
- Q** *msgkey* removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.
- M** *shmkey* removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure 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 structure associated with it.

The details of the removes are described in *msgctl(2)*, *shmctl(2)*, and *semctl(2)*. The identifiers and keys may be found by using *ipcs(1)*.

**SEE ALSO**

*ipcs(1)*, *msgctl(2)*, *msgget(2)*, *msgop(2)*, *semctl(2)*, *semget(2)*, *semop(2)*, *shmctl(2)*, *shmget(2)*, *shmop(2)*.

## NAME

`ipcs` — report inter-process communication facilities status

## SYNOPSIS

`ipcs` [ options ]

## DESCRIPTION

`Ip` prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

- q Print information about active message queues.
- m Print information about active shared memory segments.
- s Print information about active semaphores.

If any of the options —q, —m, or —s are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed.

- b Print biggest allowable 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 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* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop*(2) on semaphores.) See below.
- a Use all print *options*. (This is a shorthand notation for —b, —c, —o, —p, and —t.)
- C *corefile*  
Use the file *corefile* in place of `/dev/kmem`.
- N *namelist*  
The argument will be taken as the name of an alternate *namelist* (`/unix` 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 *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* 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.

|                |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>KEY</b>     | (all) | The key used as an argument to <i>msgget</i> , <i>semget</i> , or <i>shmget</i> to create the facility entry. (Note: The key of a shared memory segment is changed to <code>IPC_PRIVATE</code> when the segment has been removed until all processes attached to the segment detach it.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>MODE</b>    | (all) | <p>The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:</p> <p>The first two characters are:</p> <ul style="list-style-type: none"> <li><b>R</b> if a process is waiting on a <i>msgrcv</i>;</li> <li><b>S</b> if a process is waiting on a <i>msgsnd</i>;</li> <li><b>D</b> if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;</li> <li><b>C</b> if the associated shared memory segment is to be cleared when the first attach is executed;</li> <li>— if the corresponding special flag is not set.</li> </ul> <p>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.</p> <p>The permissions are indicated as follows:</p> <ul style="list-style-type: none"> <li><b>r</b> if read permission is granted;</li> <li><b>w</b> if write permission is granted;</li> <li><b>a</b> if alter permission is granted;</li> <li>— if the indicated permission is <i>not</i> granted.</li> </ul> |
| <b>OWNER</b>   | (all) | The login name of the owner of the facility entry.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>GROUP</b>   | (all) | The group name of the group of the owner of the facility entry.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>CREATOR</b> | (a,c) | The login name of the creator of the facility entry.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>CGROUP</b>  | (a,c) | The group name of the group of the creator of the facility entry.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>CBYTES</b>  | (a,o) | The number of bytes in messages currently outstanding on the associated message queue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>QNUM</b>    | (a,o) | The number of messages currently outstanding on the associated message queue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>QBYTES</b>  | (a,b) | The maximum number of bytes allowed in messages outstanding on the associated message queue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>LSPID</b>   | (a,p) | The process ID of the last process to send a message to the associated queue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

|               |       |                                                                                                     |
|---------------|-------|-----------------------------------------------------------------------------------------------------|
| <b>LRPID</b>  | (a,p) | The process ID of the last process to receive a message from the associated queue.                  |
| <b>STIME</b>  | (a,t) | The time the last message was sent to the associated queue.                                         |
| <b>RTIME</b>  | (a,t) | The time the last message was received from the associated queue.                                   |
| <b>CTIME</b>  | (a,t) | The time when the associated entry was created or changed.                                          |
| <b>NATTCH</b> | (a,o) | The number of processes attached to the associated shared memory segment.                           |
| <b>SEGSZ</b>  | (a,b) | The size of the associated shared memory segment.                                                   |
| <b>CPID</b>   | (a,p) | The process ID of the creator of the shared memory entry.                                           |
| <b>LPID</b>   | (a,p) | The process ID of the last process to attach or detach the shared memory segment.                   |
| <b>ATIME</b>  | (a,t) | The time the last attach was completed to the associated shared memory segment.                     |
| <b>DTIME</b>  | (a,t) | The time the last detach was completed on the associated shared memory segment.                     |
| <b>NSEMS</b>  | (a,b) | The number of semaphores in the set associated with the semaphore entry.                            |
| <b>OTIME</b>  | (a,t) | The time the last semaphore operation was completed on the set associated with the semaphore entry. |

**FILES**

|             |                 |
|-------------|-----------------|
| /unix       | system namelist |
| /dev/kmem   | memory          |
| /etc/passwd | user names      |
| /etc/group  | group names     |

**SEE ALSO**

msgop(2), semop(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 [ options ] file1 file2

## DESCRIPTION

*Join* forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is *-*, the standard input is used.

*File1* and *file2* 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 *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

Fields are normally separated by blank, tab or new-line. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

- an* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
- es* Replace empty output fields by string *s*.
- jn m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file.
- 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.
- tc* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

## SEE ALSO

awk(1), comm(1), sort(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* and *awk(1)* are wildly incongruous.

**NAME**

*kasb*, *kunb* — assembler/un-assembler for the KMC11B microprocessor

**SYNOPSIS**

*kasb* [ *name* ] [ *-o name1* ] [ *-d name2* ]

*kunb* [ *name* ] [ *-o name1* ]

**DESCRIPTION**

*Kasb* is an assembler/debugger/loader for the KMC11B microprocessor. The optional argument *name* specifies the input file; default is standard input. The optional argument *-o* indicates that the next argument *name1* will be the output of the assembler; default is *a.out*. The optional argument *-d* indicates that the assembler is to be used in debug mode and that the next argument *name2* is the device file name of the microprocessor. No output file is created in debug mode.

Error diagnostics are written on the standard error output and contain the input file name and line number and a brief description of the error. C preprocessor control lines to change the file name and line number are recognized. This allows the use of the preprocessor to expand the input before assembly.

*Kunb* is an un-assembler for the KMC11/DMC11 microprocessor. It produces an output listing, acceptable to the assembler *kasb*, from the input object.

The optional argument *name* specifies the input object, default is standard input. The format of the input is either assembler output (first word magic 0410), or formatted dump (first word magic 0440), or raw dump (anything else). In the first two cases, the header is ignored.

The optional argument *-o* indicates that the next argument *name1* is to contain the output listing, default is standard output.

The input object is first scanned to determine branch destinations. Labels will be inserted at these locations with format *Lint:*, where *int* is the octal value of the location in words. Immediate values of instructions are also printed in octal. Page breaks are noted by the labels *P0:*, ..., *P3:*.

**FILES**

|                  |                       |
|------------------|-----------------------|
| <i>a.out</i>     | output object         |
| <i>/dev/kmc?</i> | microprocessor device |
| <i>/lib/cpp</i>  | C preprocessor        |

**SEE ALSO**

*kmc(7)*, *vpm(7)*.

*Assembler for the DEC KMC11 Microprocessor* by L. A. Wehr.

**NAME**

kill — terminate a process

**SYNOPSIS**

kill [ -signo ] PID ...

**DESCRIPTION**

*Kill* sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with **&** is reported by the Shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by **-** is given as first argument, that signal is sent instead of terminate (see *signal*(2)). In particular “kill -9 ...” is a sure kill.

**SEE ALSO**

*ps*(1), *sh*(1), *kill*(2), *signal*(2).

## NAME

**ld** — link editor for common object files

## SYNOPSIS

**ld** [-e *epsym*] [-f *fill*] [-lx] [-m] [-r] [-s] [-o *outfile*] [-u *symname*] [-L *dir*] [-x] [-N] [-V] [-VS *num*] *file-names*

## DESCRIPTION

The *ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and *ld* combines them, producing an object module that can either be executed or used as input for a subsequent *ld* run. The output of *ld* is left in *a.out*. This file is executable if no errors occurred during the load. If any input file, *file-name*, is not an object file, *ld* assumes it is either an ASCII file containing link editor directives or an archive library.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The library (archive) symbol table (see *ar(4)*) is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is unimportant.

The following options are recognized by *ld*.

- e *epsym*  
Set the default entry point address for the output file to be that of the symbol *epsym*.
- f *fill* This option sets the default fill pattern for “holes” within an output section as well as initialized bss sections. The argument *fill* is a two-byte constant.
- lx This option specifies a library named *x*. It stands for *libx.a* where *x* is up to seven characters. A library is searched when its name is encountered, so the placement of a *-l* is significant. By default, libraries are located in */lib* and */usr/lib*.
- m This option causes a map or listing of the input/output sections to be produced on the standard output.
- o *outfile*  
This option produces an output object file by the name *outfile*. The name of the default object file is *a.out*.
- r This option causes relocation entries to be retained in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. The link editor will not complain about unresolved references.
- s This option causes line number entries and symbol table information to be stripped from the output object file.
- u *symname*  
Takes the argument *symname* as a symbol and enters it as undefined in the symbol table. This is useful for loading entirely 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 Do not preserve local (non-globl) symbols in the output symbol table; only enter external and static symbols. This option saves some space in the output file.



- L *dir*  
Change the algorithm of searching for *libx.a* to look in *dir* before looking in */lib*.
- N Put the data section immediately following the text in the output file
- V Output a message giving information about the version of *ld* being used.
- VS *num*  
The *num* argument is taken as a decimal version number identifying the *a.out* file that is produced. The version stamp is stored in the optional header.

**FILES**

|                    |             |
|--------------------|-------------|
| <i>/lib/libx.a</i> | libraries   |
| <i>a.out</i>       | output file |

**SEE ALSO**

*as(1),cc(1),a.out(4),ar(4).*

**CAVEATS**

Through its input directives, the common link editor gives users great flexibility; however, people who use the input directives must assume some added responsibilities. Input directives should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the data space.

## NAME

**ld** — link editor

## SYNOPSIS

**ld** [ **-sulxXrdnim** ] [ **-o name** ] [ **-t name** ] [ **-V num** ] file ...

## DESCRIPTION

*Ld* combines several object programs into one; resolves external references; and searches libraries (as created by *ar*(1)). In the simplest case several object files are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the **-r** option must be given to preserve the relocation bits.) The output of *ld* is left on **a.out**. This file is made executable if no errors occurred during the load and the **-r** flag was not specified.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important.

The symbols **\_etext**, **\_edata** and **\_end** (**etext**, **edata** and **end** in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

*Ld* understands several flag arguments which are written preceded by a **-**. Except for **-l**, they should appear before the file names.

- s** “Strip” the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by *strip*(1). This option is turned off if there are any undefined symbols.
- u** Take the following argument as a symbol and enter it as undefined in the symbol table. 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.
- l** This option is an abbreviation for a library name. **-l** alone stands for **/lib/libc.a**, which is the standard system library for C and assembly language programs. **-lx** stands for **/lib/libx.a**, where **x** is a string. If that does not exist, *ld* tries **/usr/lib/libx.a**. A library is searched when its name is encountered, so the placement of a **-l** is significant.
- x** Do not preserve local (non-**globl**) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.
- X** Save 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.
- 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.

- d Force definition of common storage even if the -r flag is present.
- n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 4K word boundary following the end of the text. Use -N to turn it off.
- i When the output file is executed, the program text and data areas will live in separate address spaces. The only difference between this option and -n is that here the data starts at location 0.
- m The names of all files and archive members used to create the output file are written to the standard output.
- o The *name* argument after -o is used as the name of the *ld* output file, instead of *a.out*.
- t The *name* argument is taken to be a symbol name, and any references to or definitions of that symbol are listed, along with their types. There can be up to 16 occurrences of -t*name* on the command line.
- V The *num* argument is taken as a decimal version number identifying the *a.out* that is produced. *Num* must be in the range 0-32767. The version stamp is stored in the *a.out* header; see *a.out*(4).

#### FILES

|                 |                |
|-----------------|----------------|
| /lib/lib?.a     | libraries      |
| /usr/lib/lib?.a | more libraries |
| a.out           | output file    |

#### SEE ALSO

ar(1), as(1), cc(1), a.out(4), ar(4).

**NAME**

`lex` — generate programs for simple lexical tasks

**SYNOPSIS**

`lex [ -rctvn ] [ file ] ...`

**DESCRIPTION**

*Lex* generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file `lex.yy.c` is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. 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 brackets to indicate character classes, as in `[abx-z]` to indicate `a`, `b`, `x`, `y`, and `z`; and the operators `*`, `+`, and `?` mean respectively any non-negative number of, any positive number of, and either zero or one occurrences of, the previous character or character class. The character `.` is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported. The notation `r{d,e}` in a rule indicates between `d` and `e` instances of regular expression `r`. It has higher precedence than `|`, but lower than `*`, `?`, `+`, and concatenation. The character `^` at the beginning of an expression permits a successful match only immediately after a new-line, and the character `$` at the end of an expression requires a trailing new-line. The character `/` in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but 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 `\`. Thus `[a-zA-Z]+` matches a string of letters.

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 causes this match to be rejected and the next suitable match executed; the function `yyomore()` accumulates additional characters into the same *yytext*; and the function `yyless(p)` pushes back the portion of the string matched beginning at `p`, which should be between *yytext* and `yytext+yy leng`. The macros `input` and `output` use files `yyin` and `yyout` to read from and write to, defaulted to `stdin` and `stdout`, respectively.

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 non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with `{}`. Note that curly brackets do not imply parentheses; only string substitution is done.

**EXAMPLE**

```
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;
        }
    }

```

The external names generated by *lex* all begin with the prefix *yy* or *YY*.

The flags must appear before any files. The flag *-r* indicates RATFOR actions, *-c* indicates C actions and is the default, *-t* causes the *lex.yy.c* program to be written instead to standard output, *-v* provides a one-line summary of statistics of the machine generated, *-n* will not print out the *-* summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

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.

#### SEE ALSO

*yacc*(1).

*LEX—Lexical Analyzer Generator* by M. E. Lesk and E. Schmidt.

#### BUGS

The *-r* option is not yet fully operational.

**NAME**

line — read one line

**SYNOPSIS**

line

**DESCRIPTION**

*Line* copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

**SEE ALSO**

sh(1), read(2).

**NAME**

`lint` — a C program checker

**SYNOPSIS**

`lint [ -abhlmpvx ] file ...`

**DESCRIPTION**

*Lint* attempts to detect features of the C program *files* which are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things which are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

It is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. By default, *lint* uses function definitions from the standard *lint* library `llib-lc.ln`; function definitions from the portable *lint* library `llib-port.ln` are used when *lint* is invoked with the `-p` option.

Any number of *lint* options may be used, in any order. The following options are used to suppress certain kinds of complaints:

- a Suppress complaints about assignments of long values to variables that are not long.
- b Suppress complaints about `break` statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in a large number of such complaints.)
- h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.
- u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program.)
- v Suppress complaints about unused arguments in functions.
- x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint's* behavior:

- lx Include additional *lint* library `llib-lx.ln`. You can include a *lint* version of the math library `llib-lm.ln` by inserting `-lm` on the command line. This argument does not suppress the default use of `llib-lc.ln`. This option can be used to keep local *lint* libraries and is useful in the development of multi-file projects.
- n Do not check compatibility against either the standard or the portable *lint* library.
- p Attempt to check portability to other dialects (IBM and GCOS) of C.

The `-D`, `-U`, and `-I` options of `cc(1)` are also recognized as separate arguments.

Certain conventional comments in the C source will change the behavior of *lint*:

```
/*NOTREACHED*/
    at appropriate points stops comments about unreachable
    code.
```

**/\*VARARGS*n*\*/**

suppresses 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.

**/\*ARGSUSED\*/**

turns on the **-v** option for the next function.

**/\*LINTLIBRARY\*/**

at the beginning of a file shuts off complaints about unused functions in this file.

*Lint* produces its first output on a per source file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, information gathered from all input files is 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 will be printed followed by a question mark.

**FILES**

|                                     |                                                                                                        |
|-------------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>/usr/lib/lint[12]</code>      | programs                                                                                               |
| <code>/usr/lib/l-lib-1c.ln</code>   | declarations for standard functions (binary format; source is in <code>/usr/lib/l-lib-1c</code> )      |
| <code>/usr/lib/l-lib-port.ln</code> | declarations for portable functions (binary format; source is in <code>/usr/lib/l-lib-port</code> )    |
| <code>/usr/lib/l-lib-1m.ln</code>   | declarations for standard math functions (binary format; source is in <code>/usr/lib/l-lib-1m</code> ) |
| <code>/usr/tmp/*lint*</code>        | temporaries                                                                                            |

**SEE ALSO**

`cc(1)`.

**BUGS**

*Exit(2)* and other functions which do not return are not understood; this causes various lies.



**NAME**

*list* — produce C source listing from 3B20S object file

**SYNOPSIS**

*list* [ *-V* ] [ *-h* ] source-file . . . [object-file]

**DESCRIPTION**

The *list* command produces a C source listing with line number information attached. If multiple C source files were used to create the object file, *list* will accept multiple file names. The object file is taken to be the last non-C source file argument. If no object file is specified the default object file, *a.out*, will be used.

Line numbers will be printed for each breakpoint inserted by the compiler (generally, each executable C statement that begins a new line of source). Line numbering begins anew for each function. Line number 1 is always the line containing the left curly brace ( { ) that begins the function body. Line numbers will also be supplied for inner block redeclarations of local variables so that they can be distinguished by the symbolic debugger.

The *-V* flag will supply version information of the *list* command.

The *-h* flag will suppress heading output.

**CAVEATS**

Object files given to *list* must have symbolic debugging symbols.

Since *list* does not use the C preprocessor, it may be unable to recognize function definitions whose syntax has been distorted by the use of C preprocessor macro substitutions.

**SEE ALSO**

*as*(1), *cc*(1), *ld*(1).

**DIAGNOSTICS**

“*list: name: cannot open*” if *name* cannot be read.

## NAME

login — sign on

## SYNOPSIS

login [ name [ env-var ... ] ]

## DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an “end-of-file.” (See *How to Get Started* at the beginning of this volume for instructions on how to dial up initially.)

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

```
exec login
from the initial shell.
```

*Login* asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second “dialup” password. This will occur only for dial-up connections, and will be prompted by the message “dialup password:”. Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure */etc/profile* is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh*(1)) is initialized, and the file *.profile* in the working directory is executed, if it exists. These specifications are found in the */etc/passwd* file entry for the user. The name of the command interpreter is — followed by the last component of the interpreter’s pathname (i.e., *-sh*). If this field in the password file is empty, then the default command interpreter, */bin/sh* is used.

The basic *environment* (see *environ*(5)) is initialized to:

```
HOME=your-login-directory
PATH=:/bin:/usr/bin
SHELL=last-field-of-passwd-entry
MAIL=/usr/mail/your-login-name
TZ=timezone-specification
```

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your login name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

```
Ln=xxx
```

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables *PATH* and *SHELL* cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells

which aren't restricted. Both *login* and *getty* understand simple single character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

#### FILES

|                            |                                   |
|----------------------------|-----------------------------------|
| <i>/etc/utmp</i>           | accounting                        |
| <i>/etc/wtmp</i>           | accounting                        |
| <i>/usr/mail/your-name</i> | mailbox for user <i>your-name</i> |
| <i>/etc/motd</i>           | message-of-the-day                |
| <i>/etc/passwd</i>         | password file                     |
| <i>/etc/profile</i>        | system profile                    |
| <i>.profile</i>            | user's login profile              |

#### SEE ALSO

*mail(1)*, *newgrp(1)*, *sh(1)*, *su(1)*, *passwd(4)*, *profile(4)*, *environ(5)*.

#### DIAGNOSTICS

*Login incorrect* if the user name or the password cannot be matched.

*No shell, cannot open password file, or no directory:* consult a UNIX programming counselor.

*No utmp entry. You must exec "login" from the lowest level "sh".* if you attempted to execute *login* as a command, without using the shell's *exec* internal command or from other than the initial shell.

**LOGNAME(1)**

**LOGNAME(1)**

**NAME**

logname — get login name

**SYNOPSIS**

**logname**

**DESCRIPTION**

*Logname* returns the contents of the environment variable **SLOGNAME**, which is set when a user logs into the system.

**FILES**

/etc/profile

**SEE ALSO**

env(1), login(1), logname(3X), environ(5).

**NAME**

lorder — find ordering relation for an object library

**SYNOPSIS**

lorder file ...

**DESCRIPTION**

The input is one or more object or library archive *files* (see *ar(1)*). The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort(1)* to find an ordering of a library suitable for one-pass access by *ld(1)*. Note that the link editor (except on the PDP -11) *ld(1)* is capable of multiple passes over an archive in the portable archive format (see *ar(4)*) and does not require that *lorder(1)* be used when building an archive. The usage of the *lorder(1)* command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing *.o* files.

```
ar cr library `lorder *.o | tsort`
```

**FILES**

\*symref, \*symdef      temporary files

**SEE ALSO**

*ar(1)*, *ld(1)*, *tsort(1)*, *ar(4)*.

**BUGS**

Object files whose names do not end with *.o*, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

## NAME

*lp*, *cancel* — send/cancel requests to an LP line printer

## SYNOPSIS

```
lp [-c] [-ddest] [-m] [-nnumber] [-ooption] [-s] [-ttitle] [-w]
files
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 *files*. The order in which *files* 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.

The following options to *lp* may appear in any order and may be intermixed with file names:

- c            Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* 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 *files* 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 *files* after the request is made but before it is printed will be reflected in the printed output.
- d*dest*      Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see *accept*(1M) and *lpstat*(1)). By default, *dest* is taken from the environment variable LPDEST (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).
- 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.
- n*number*    Print *number* copies (default of 1) of the output.
- o*option*     Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the *-o* keyletter more than once. For more information about what is valid for *options*, see *Models* in *lpadmin*(1M).
- s            Suppress messages from *lp*(1) such as "request id is ...".
- t*title*      Print *title* on the banner page of the output.
- w            Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

*Cancel* cancels line printer requests that were made by the *lp(1)* command. The command line arguments may be either request *ids* (as returned by *lp(1)*) or *printer* names (for a complete list, 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.

**FILES**

/usr/spool/lp/\*

**SEE ALSO**

*enable(1)*, *lpstat(1)*, *mail(1)*.  
*accept(1M)*, *lpadmin(1M)*, *lpsched(1M)* in the *UNIX System Administrator's Manual*.

**NAME**

`lpr` — line printer spooler

**SYNOPSIS**

`lpr` [ option ... ] [ name ... ]

**DESCRIPTION**

*Lpr* causes the named files to be queued for printing on a line printer. If no names appear, the standard input is assumed; thus *lpr* may be used as a filter.

The following *options* may be given (each as a separate argument and in any order) before any file name arguments:

- c Makes a copy of the file to be sent before returning to the user.
- r Removes the file after sending it.
- m When printing is complete, reports that fact by *mail*(1).
- n Does not report the completion of printing by *mail*(1). This is the default option.
- f*file* Use *file* as a dummy file name to report back in the mail. (This is useful for distinguishing multiple runs, especially when *lpr* is being used as a filter).

**FILES**

|                               |                                            |
|-------------------------------|--------------------------------------------|
| <code>/etc/passwd</code>      | user's identification and accounting data. |
| <code>/usr/lib/lpd</code>     | line printer daemon.                       |
| <code>/usr/spool/lpd/*</code> | spool area.                                |

**SEE ALSO**

`dpd`(1C), `dpr`(1C), `lp`(1).



## NAME

`lpstat` — print LP status information

## SYNOPSIS

`lpstat` [*options*]

## DESCRIPTION

*Lpstat* prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *Lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters 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 spaces. For example:

`-u"user1, user2, user3"`

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

`lpstat -o`

prints the status of all output requests.

- `-a[list]` Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.
- `-c[list]` Print class names and their members. *List* is a list of class names.
- `-d` Print the system default destination for *lp*.
- `-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.
- `-r` Print the status of the LP 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.
- `-u[list]` Print status of output requests for users. *List* is a list of login names.
- `-v[list]` Print the names of printers and the pathnames of the devices associated with them. *List* is a list of printer names.

## FILES

`/usr/spool/lp/*`

## SEE ALSO

`enable`(1), `lp`(1).

**NAME**

`ls` — list contents of directories

**SYNOPSIS**

`ls [ -logtasdrucifp ] names`

**DESCRIPTION**

For each directory named, `ls` lists the contents of that directory; for each file named, `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. There are several options:

- l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will contain the major and minor device numbers, rather than a size.
- o The same as -l, except that the group is not printed.
- g The same as -l, except that the owner is not printed.
- t Sort by time of last modification (latest first) instead of by name.
- a List all entries; in the absence of this option, entries whose names begin with a period (.) are *not* listed.
- s Give size in blocks (including indirect blocks) for each entry.
- d If argument is a directory, list only its name; often used with -l to get the status of a directory.
- r Reverse the order of sort to get reverse alphabetic or oldest first, as appropriate.
- u Use time of last access instead of last modification for sorting (with the -t option) and/or printing (with the -l option).
- c Use time of last modification of the inode (mode, etc.) instead of last modification of the file for sorting (-t) and/or printing (-l).
- i For each file, print the i-number in the first column of the report.
- 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.
- p Put a slash after each filename if that file is a directory. Especially useful for CRT terminals when combined with the `pr(1)` command as follows: `ls -p | pr -5 -t -w80`.

The mode printed under the -l option consists of 11 characters that are interpreted as follows:

The first character is:

- d** if the entry is a directory;
- b** if the entry is a block special file;
- c** if the entry is a character special file;
- p** if the entry is a fifo (a.k.a. "named pipe") special file;
- if the entry is an ordinary file.

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 file; and the last to

all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, “execute” permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

- r** if the file is readable;
- w** if the file is writable;
- x** if the file is executable;
- if the indicated permission is *not* granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user-execute permission character is given as **S** if the file has set-user-ID mode. The last character of the mode (normally **x** or **—**) is **t** if the 1000 (octal) bit of the mode is on; see *chmod(1)* for the meaning of this mode. The indications of set-ID and 1000 bit 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.

#### FILES

`/etc/passwd` to get user IDs for `ls -l` and `ls -o`.  
`/etc/group` to get group IDs for `ls -l` and `ls -g`.

#### SEE ALSO

`chmod(1)`, `find(1)`.

## NAME

m4 — macro processor

## SYNOPSIS

m4 [ options ] [ files ]

## 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.

The options and their effects are as follows:

- e Operate interactively. Interrupts are ignored and the output is unbuffered. Using this mode requires a special state of mind.
- 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]  
Defines *name* to *val* or to null in *val*'s absence.
- U*name*  
undefines *name*.

Macro calls have the form:

name(arg1,arg2, ..., argn)

The `(` must immediately follow the name of the macro. If the name of a defined macro is not followed by a `(`, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore `_`, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines 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, its 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.

*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;  $\$#$  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).

- undefine    removes the definition of the macro named in its argument.
- defn        returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.
- pushdef    like *define*, but saves any previous definition.
- popdef     removes current definition of its argument(s), exposing the previous one if any.
- 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 on UNIX versions of *m4*.
- shift       returns all but its 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.
- changequote    change quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., ``'').
- changecom    change left and right comment markers from the default  $\#$  and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.
- 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 its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
- undivert     causes 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       returns the value of the current output stream.
- dnl          reads and discards characters up to and including the next new-line.
- 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 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
- incr         returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>decr</code>     | returns the value of its argument decremented by 1.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <code>eval</code>     | evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> , <code>^</code> (exponentiation), bitwise <code>&amp;</code> , <code> </code> , <code>^</code> , and <code>~</code> ; relationals; parentheses. Octal 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. |
| <code>len</code>      | returns the number of characters in its argument.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <code>index</code>    | returns the position in its first argument where the second argument begins (zero origin), or <code>-1</code> if the second argument does not occur.                                                                                                                                                                                                                                                                                                                                                                                                |
| <code>substr</code>   | returns a substring of its 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.                                                                                                                                                                                                                                                                      |
| <code>translit</code> | transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.                                                                                                                                                                                                                                                                                                                                                                                        |
| <code>include</code>  | returns the contents of the file named in the argument.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>sinclude</code> | is identical to <i>include</i> , except that it says nothing if the file is inaccessible.                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <code>syscmd</code>   | executes the UNIX command given in the first argument. No value is returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <code>sysval</code>   | is the return code from the last call to <i>syscmd</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <code>maketemp</code> | fills in a string of <code>XXXXX</code> in its argument with the current process ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <code>m4exit</code>   | causes immediate exit from <i>m4</i> . Argument 1, if given, is the exit code; the default is 0.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <code>m4wrap</code>   | argument 1 will be pushed back at final EOF; example: <code>m4wrap(^cleanup(^)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <code>errprint</code> | prints its argument on the diagnostic output file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <code>dumpdef</code>  | prints current names and definitions, for the named items, or for all if no arguments are given.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <code>traceon</code>  | with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>traceoff</code> | turns off trace globally and for any macros specified. Macros specifically traced by <i>traceon</i> can be untraced only by specific calls to <i>traceoff</i> .                                                                                                                                                                                                                                                                                                                                                                                     |

## SEE ALSO

`cc(1)`, `cpp(1)`. *The M4 Macro Processor* by B. W. Kernighan and D. M. Ritchie.

**NAME**

pdp11, u3b, u3b5, vax — provide truth value about your processor type

**SYNOPSIS**

**pdp11**

**u3b**

**u3b5**

**vax**

**DESCRIPTION**

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

**pdp11** True if you are on a PDP-11/45 or PDP-11/70.

**u3b** True if you are on a 3B20S.

**u3b5** True if you are on a 3B5.

**vax** True if you are on a VAX-11/750 or VAX-11/780.

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**

sh(1), test(1), true(1).

## NAME

mail, rmail - send mail to users or read mail

## SYNOPSIS

mail [ -epqr ] [ -f file ]

mail [ -t ] persons

rmail [ -t ] persons

## DESCRIPTION

*Mail* without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a `?`, and a line is read from the standard input to determine the disposition of the message:

|                 |                                                                                        |
|-----------------|----------------------------------------------------------------------------------------|
| <new-line>      | Go on to next message.                                                                 |
| +               | Same as <new-line>.                                                                    |
| d               | Delete message and go on to next message.                                              |
| p               | Print message again.                                                                   |
| -               | Go back to previous message.                                                           |
| s [ files ]     | Save message in the named <i>files</i> ( <i>mbox</i> is default).                      |
| w [ files ]     | Save message, without its header, in the named <i>files</i> ( <i>mbox</i> is default). |
| m [ persons ]   | Mail the message to the named <i>persons</i> (yourself is default).                    |
| q               | Put undeleted mail back in the <i>mailfile</i> and stop.                               |
| EOT (control-d) | Same as <i>q</i> .                                                                     |
| x               | Put all mail back in the <i>mailfile</i> unchanged and stop.                           |
| !command        | Escape to the shell to do <i>command</i> .                                             |
| *               | Print a command summary.                                                               |

The optional arguments alter the printing of the mail:

- e causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.
- p causes all mail to be printed without prompting for disposition.
- q causes *mail* to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.
- r causes messages to be printed in first-in, first-out order.
- f*file* causes *mail* to use *file* (e.g., *mbox*) instead of the default *mailfile*.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a `.`) and adds it to each *person's* *mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., "From ...") are preceded with a `>`. The `-t` option causes the message to be preceded by all *persons* the *mail* is sent to. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file *dead.letter* will be saved to allow editing and resending.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)). Everything after the first exclamation mark in *persons* is interpreted by the remote system. In particular, if *persons* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying *a!b!cde* as a recipient's name causes the message to be sent to user *b!cde* on system *a*. System *a* will interpret that destination as a request to send the message to user *cde* on



system **b**. This might be useful, for instance, if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**.

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment.

*Rmail* only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

#### FILES

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>/etc/passwd</i>      | to identify sender and locate persons                     |
| <i>/usr/mail/user</i>   | incoming mail for <i>user</i> ; i.e., the <i>mailfile</i> |
| <i>\$HOME/mbox</i>      | saved mail                                                |
| <i>\$MAIL</i>           | variable containing path name of <i>mailfile</i>          |
| <i>/tmp/ma*</i>         | temporary file                                            |
| <i>/usr/mail/*.lock</i> | lock for mail directory                                   |
| <i>dead.letter</i>      | unmailable text                                           |

#### SEE ALSO

*login*(1), *uucp*(1C), *write*(1).

#### BUGS

Race conditions sometimes result in a failure to remove a lock file. After an interrupt, the next message may not be printed; printing may be forced by typing a *p*.

## NAME

make — maintain, update, and regenerate groups of programs

## SYNOPSIS

make [-f *makefile*] [-p] [-i] [-k] [-s] [-r] [-n] [-b] [-e] [-m] [-t] [-d] [-q] [names]

## DESCRIPTION

The following is a brief description of all options and some special names:

- f *makefile* Description file name. *Makefile* is assumed to be the name of a description file. A file name of — denotes the standard input. The contents of *makefile* override the built-in rules if they are present.
- p Print out the complete set of macro definitions and target descriptions.
- i Ignore error codes returned by invoked commands. This mode is entered if the fake target name .IGNORE appears in the description file.
- k Abandon work on the current entry, but continue on other branches that do not depend on that entry.
- s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name .SILENT appears in the description file.
- r Do not use the built-in rules.
- n No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.
- b Compatibility mode for old makefiles.
- e Environment variables override assignments within makefiles.
- m Print a memory map showing text, data, and stack. This option is a no-operation on systems without the *getu* system call.
- t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.
- d Debug mode. Print out detailed information on files and times examined.
- q Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.
- .DEFAULT If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name .DEFAULT are used if it exists.
- .PRECIOUS Dependents of this target will not be removed when quit or interrupt are hit.
- .SILENT Same effect as the -s option.
- .IGNORE Same effect as the -i option.

*Make* executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no -f option is present, *makefile*, *Makefile*, *s.makefile*, and *s.Makefile* are tried in order. If *makefile* is —, the standard input is taken. More than one — *makefile* argument pair may appear.

*Make* updates a target only if it depends on files that are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out of date.

*Makefile* contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a `;`, then a (possibly null) list of prerequisite files or dependencies. Text following a `;` and all following lines that begin with a tab are shell commands to be executed to update the target. The first line that does not begin with a tab or `#` begins a new dependency or macro definition. Shell commands may be continued across lines with the `<backslash><new-line>` sequence. Everything printed by *make* (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\  
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (`#`) and new-line surround comments.

The following *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`) and a common file `incl.h`:

```
pgm: a.o b.o  
    cc a.o b.o -o pgm  
a.o: incl.h a.c  
    cc -c a.c  
b.o: incl.h b.c  
    cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The first one or two characters in a command can be the following: `-`, `@`, `-@`, or `@-`. If `@` is present, printing of the command is suppressed. If `-` is present, *make* ignores an error. A line is printed when it is executed unless the `-s` option is present, or the entry `.SILENT:` is in *makefile*, or unless the initial character sequence contains a `@`. The `-n` option specifies printing without execution; however, if the command line has the string `$(MAKE)` in it, the line is always executed (see discussion of the `MAKEFLAGS` macro under *Environment*). The `-t` (`touch`) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the `-i` option is present, or the entry `.IGNORE:` appears in *makefile*, or the initial character sequence of the command contains `-`, the error is ignored. If the `-k` option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The `-b` option allows old *makefiles* (those written for the old version of *make*) to run without errors. The difference between the old version of *make* and this version is that this version requires all dependency lines to have a (possibly null or implicit) command associated with them. The previous version of *make* assumed if no command was specified explicitly that the command was null.

Interrupt and quit cause the target to be deleted unless the target is a dependency of the special name `.PRECIOUS`.

## Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The `-e` option causes the environment to override the macro assignments in a makefile.

The `MAKEFLAGS` environment variable is processed by *make* as containing any legal input option (except `-f`, `-p`, and `-d`) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, `MAKEFLAGS` always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the `-n` option is used, the command `$(MAKE)` is executed anyway; hence, one can perform a `make -n` recursively on a whole software system to see what would have been executed. This is because the `-n` is put in `MAKEFLAGS` and passed to further invocations of `$(MAKE)`. This is one way of debugging all of the makefiles for a software project without actually doing anything.

## Macros

Entries of the form `string1 = string2` are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of `$(string1[:subst1=[subst2]])` are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional `:subst1=subst2` is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

## Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

- \$\*** The macro `$*` stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.
- \$@** The `$@` macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.
- \$<** The `$<` macro is only evaluated for inference rules or the `.DEFAULT` rule. It is the module which is out of date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the `.c.o` rule, the `$<` macro would evaluate to the `.c` file. An example for making optimized `.o` files from `.c` files is:

```
.c.o:
    cc -c -O $*.c
```

or:

```
.c.o:
    cc -c -O $<
```

- \$?** The `$?` macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out of date with respect to the target; essentially, those modules which must be rebuilt.
- \$%** The `$%` 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`.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus,  $\$(@D)$  refers to the directory part of the string  $\$@$ . If there is no directory part,  $./$  is generated. The only macro excluded from this alternative form is  $\$?$ . The reasons for this are debatable.

### Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c~ .sh .sh~ .c.o .c~.o .c~.c .s.o .s~.o .y.o .y~.o .l.o .l~.o
.y.c .y~.c .l.c .c.a .c~.a .s~.a .h~.h
```

The internal rules for *make* are contained in the source file *rules.c* for the *make* program. These rules can be locally modified. To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
make -fp - 2>/dev/null </dev/null
```

The only peculiarity in this output is the (**null**) string which *printf(3S)* prints when handed a null string.

A tilde in the above rules refers to an SCCS file (see *sccsfile(4)*). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix it is incompatible with *make*'s suffix point-of-view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e. **.c**) is the definition of how to build *x* from *x.c*. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

```
.SUFFIXES: .o .c .y .l .s
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

### Inference Rules

The first example can be done more briefly:

```
pgm: a.o b.o
      cc a.o b.o -o pgm
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc(1)*, *lex(1)*, and *yacc(1)* respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix `.o` from a file with suffix `.c` is specified as an entry with `.c.o`: as the target and no dependents. Shell commands associated with the target define the rule for making a `.o` file from a `.c` file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

### Libraries

If a target or dependency name contains parenthesis, it is assumed to be an archive library, the string within parenthesis referring to a member within the library. Thus `lib(file.o)` and `$(LIB)(file.o)` both refer to an archive library which contains `file.o`. (This assumes the `LIB` macro has been previously defined.) The expression `$(LIB)(file1.o file2.o)` is not legal. 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 the `XX` to be different from the suffix of the archive member. Thus, one cannot have `lib(file.o)` depend upon `file.o` explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up to date

.c.a:
        $(CC) -c $(CFLAGS) $<
        ar rv $@ $*.o
        rm -f $*.o
```

In fact, the `.c.a` rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        ar rv lib $?
        rm $? @echo lib is now up to date

.c.a;;
```

Here the substitution mode of the macro expansions is used. The `?$` list is defined to be the set of object file names (inside `lib`) whose C source files are out of date. The substitution mode translates the `.o` to `.c`. (Unfortunately, one cannot as yet transform to `.c`; however, this may become possible in the future.) Note also, the disabling of the `.c.a`: rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

### FILES

[Mm]akefile and s.[Mm]akefile

### SEE ALSO

`sh(1)`.

*Make—A Program for Maintaining Computer Programs* by S. I. Feldman.  
*An Augmented Version of Make* by E. G. Bradford.

### BUGS

Some commands return non-zero status inappropriately; use `-i` to overcome the difficulty. Commands that are directly executed by the shell, notably `cd(1)`, are ineffectual across new-lines in *make*. The syntax (`lib(file1.o file2.o file3.o)`) is illegal. You cannot build `lib(file.o)` from `file.o`. The macro `$(a:.o=.c)` doesn't work.

**NAME**

makekey — generate encryption key

**SYNOPSIS**

/usr/lib/makekey

**DESCRIPTION**

*Makekey* improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

*Makekey* is intended for programs that perform encryption (e.g., *ed*(1) and *crypt*(1)). Usually, its input and output will be pipes.

**SEE ALSO**

*crypt*(1), *ed*(1), *passwd*(4).

## NAME

man, manprog — print entries in this manual

## SYNOPSIS

**man** [ options ] [ section ] titles  
 /usr/lib/manprog file

## DESCRIPTION

*Man* locates and prints the entry of this manual named *title* in the specified *section*. (For historical reasons, the word “page” is often used as a synonym for “entry” in this context.) The *title* is entered in lower case. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for *title* and all occurrences of it are printed. *Options* and their meanings are:

- t           Typeset the entry in the default format (8.5"×11").
- s           Typeset the entry in the small format (6"×9").
- T4014       Display the typeset output on a Tektronix 4014 terminal using *tc*(1).
- Ttek        Same as —T4014.
- Tst         Print the typeset output on the MHCC STARE facility (this option is not usable on most systems).
- Tvp         Print the typeset output on a Versatec printer; this option is not available at all UNIX sites.
- Tterm       Format the entry using *nroff* and print it on the standard output (usually, the terminal); *term* is the terminal type (see *term*(5) and the explanation below); for a list of recognized values of *term*, type *help term2*. The default value of *term* is 450.
- w           Print on the standard output only the *path names* of the entries, relative to /usr/man, or to the current directory for —d option.
- d           Search the current directory rather than /usr/man; requires the full file name (e.g., *cu.1c*, rather than just *cu*).
- 12          Indicates that the manual entry is to be produced in 12-pitch. May be used when \$TERM (see below) is set to one of 300, 300s, 450, and 1620. (The pitch switch on the DASI 300 and 300s terminals must be manually set to 12 if this option is used.)
- c           Causes *man* to invoke *col*(1); note that *col*(1) is invoked automatically by *man* unless *term* is one of 300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620, and X.
- y           Causes *man* to use the non-compacted version of the macros.

The above *options* other than —d, —c, and —y are mutually exclusive, except that the —s option may be used in conjunction with the first four —T options above. Any other *options* are passed to *troff*, *nroff*, or the *man*(5) macro package.

When using *nroff*, *man* examines the environment variable \$TERM (see *environ*(5)) and attempts to select options to *nroff*, as well as filters, that adapt the output to the terminal being used. The —Tterm option overrides the value of \$TERM; in particular, one should use —Tlp when sending the output of *man* to a line printer.

*Section* may be changed before each *title*.

As an example:

```
man man
```

would reproduce on the terminal this entry, as well as any other entries named *man* that may exist in other sections of the manual, e.g., *man*(5).



If the first line of the input for an entry consists solely of the string:

```
\* x
```

where *x* is any combination of the three characters *c*, *e*, and *t*, and where there is exactly one blank between the double quote (") and *x*, then *man* will preprocess its input through the appropriate combination of *cw*(1), *eqn*(1) (*neqn* for *nroff*) and *tbl*(1), respectively; if *eqn* or *neqn* are invoked, they will automatically read the file */usr/pub/eqnchar* (see *eqnchar*(5)).

The *man* command executes *manprog* that takes a file name as its argument. *Manprog* calculates and returns a string of three register definitions used by the formatters identifying the date the file was last modified. The returned string has the form:

```
-rdday -rmmonth -ryyear
```

and is passed to *nroff* which sets this string as variables for the *man* macro package. Months are given from 0 to 11, therefore month is always 1 less than the actual month. The *man* macros calculate the correct month. If the *man* macro package is invoked as an option to *nroff/troff* (i.e., *nroff -man file*), then the current day/month/year is used as the printed date.

#### FILES

|                                  |                                               |
|----------------------------------|-----------------------------------------------|
| <i>/usr/man/u_man/man[1-6]/*</i> | the <i>UNIX System User's Manual</i>          |
| <i>/usr/man/a_man/man[178]/*</i> | the <i>UNIX System Administrator's Manual</i> |
| <i>/usr/man/local/man[1-8]/*</i> | local additions                               |
| <i>/usr/lib/manprog</i>          | calculates modification dates of entries      |

#### SEE ALSO

*cw*(1), *eqn*(1), *nroff*(1), *tbl*(1), *tc*(1), *troff*(1), *environ*(5), *man*(5), *term*(5).

#### BUGS

All entries are supposed to be reproducible either on a typesetter or on a terminal. However, on a terminal some information is necessarily lost.

Pages bearing the same name in both manuals will result in the *UNIX System Administrator's Manual* entry being printed first, if no *section* argument is supplied.

**NAME**

**msg** — permit or deny messages

**SYNOPSIS**

**msg** [ **n** ] [ **y** ]

**DESCRIPTION**

*Msg* with argument **n** forbids messages via *write(1)* by revoking non-user write permission on the user's terminal. *Msg* with argument **y** reinstates permission. All by itself, *msg* reports the current state without changing it.

**FILES**

/dev/tty\*

**SEE ALSO**

*write(1)*.

**DIAGNOSTICS**

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

**NAME**

**mkdir** -- make a directory

**SYNOPSIS**

**mkdir** dirname ...

**DESCRIPTION**

*Mkdir* creates specified directories in mode 777 (possibly altered by *umask*(1)). Standard entries, *.*, for the directory itself, and *..*, for its parent, are made automatically.

*Mkdir* requires write permission in the parent directory.

**SEE ALSO**

*sh*(1), *rm*(1), *umask*(1).

**DIAGNOSTICS**

*Mkdir* returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

## NAME

**mm**, **osdd**, **checkmm** — print/check documents formatted with the MM macros

## SYNOPSIS

**mm** [ options ] [ files ]

**osdd** [ options ] [ files ]

**checkmm** [ files ]

## DESCRIPTION

*Mm* can be used to type out documents using *nroff* and the MM text-formatting macro package. It has options to specify preprocessing by *tbl(1)* and/or *neqn* (see *eqn(1)*) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff* and MM are generated, depending on the options selected.

*Osdd* is equivalent to the command **mm -mosd**. For more information about the OSDD adapter macro package, see *mosd(5)*.

*Options* for *mm* are given below. Any other arguments or flags (e.g., **-rC3**) are passed to *nroff* or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

- Term** Specifies the type of output terminal; for a list of recognized values for *term*, type **help term2**. If this option is *not* used, *mm* will use the value of the shell variable **\$TERM** from the environment (see *profile(4)* and *environ(5)*) as the value of *term*, if **\$TERM** is set; otherwise, *mm* will use **450** as the value of *term*. If several terminal types are specified, the last one takes precedence.
- 12** Indicates that the document is to be produced in 12-pitch. May be used when **\$TERM** is set to one of **300**, **300s**, **450**, and **1620**. (The pitch switch on the DASI 300 and 300s terminals must be manually set to **12** if this option is used.)
- c** Causes *mm* to invoke *col(1)*; note that *col(1)* is invoked automatically by *mm* unless *term* is one of **300**, **300s**, **450**, **37**, **4000a**, **382**, **4014**, **tek**, **1620**, and **X**.
- e** Causes *mm* to invoke *neqn*; also causes *neqn* to read the **/usr/pub/eqnchar** file (see *eqnchar(5)*).
- t** Causes *mm* to invoke *tbl(1)*.
- E** Invokes the **-e** option of *nroff*.
- y** Causes *mm* to use the non-compacted version of the macros (see *mm(5)*).

As an example (assuming that the shell variable **\$TERM** is set in the environment to **450**), the two command lines below are equivalent:

```
mm -t -rC3 -12 ghh*
tbl ghh* | nroff -cm -T450-12 -h -rC3
```

*Mm* reads the standard input when **-** is specified instead of any file names. (Mentioning other files together with **-** leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

```
cat dws | mm -
```

*Checkmm* is a program for checking the contents of the named *files* for errors in the use of the Memorandum Macros, missing or unbalanced *neqn* delimiters, and **.EQ/.EN** pairs. Note: The user need not use the *checkeq* program (see *eqn(1)*). Appropriate messages are produced. The program skips all directories, and if no file name is given, standard input is read.

## HINTS

1. *Mm* invokes *nroff* with the *-h* flag. With this flag, *nroff* assumes that the terminal has tabs set every 8 character positions.
2. Use the *-olist* option of *nroff* to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of the *-e*, *-t*, and *-* options, together with the *-olist* option of *nroff* may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.
3. If you use the *-s* option of *nroff* (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The *-s* option of *nroff* does not work with the *-c* option of *mm*, or if *mm* automatically invokes *col(1)* (see *-c* option above).
4. If you lie to *mm* about the kind of terminal its output will be printed on, you'll get (often subtle) garbage; however, if you are redirecting output into a file, use the *-T37* option, and then use the appropriate terminal filter when you actually print that file.

## SEE ALSO

*col(1)*, *cw(1)*, *env(1)*, *eqn(1)*, *greek(1)*, *mmt(1)*, *nroff(1)*, *tbl(1)*, *profile(4)*, *mm(5)*, *mosd(5)*, *term(5)*.

*UNIX System Document Processing Guide*.

## DIAGNOSTICS

- mm* "mm: no input file" if none of the arguments is a readable file and *mm* is not used as a filter.
- checkmm* "Cannot open filename" if file(s) is unreadable. The remaining output of the program is diagnostic of the source file.

## NAME

*mmt*, *mvt* — typeset documents, view graphs, and slides

## SYNOPSIS

*mmt* [ options ] [ files ]

*mvt* [ options ] [ files ]

## DESCRIPTION

These two commands are very similar to *mm*(1), except that they both typeset their input via *troff*(1), as opposed to formatting it via *nroff*; *mmt* uses the MM macro package, while *mvt* uses the Macro Package for View Graphs and Slides. These two commands have options to specify preprocessing by *tbl*(1) and/or *eqn*(1). The proper pipelines and the required arguments and flags for *troff*(1) and for the macro packages are generated, depending on the options selected.

*Options* are given below. Any other arguments or flags (e.g., *-rC3*) are passed to *troff*(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, these commands print a list of their options.

- e* Causes these commands to invoke *eqn*(1); also causes *eqn* to read the */usr/pub/eqnchar* file (see *eqnchar*(5)).
- t* Causes these commands to invoke *tbl*(1).
- Tst* Directs the output to the MH STARE facility.
- Tvp* Directs the output to a Versatec printer; this option is not available at all UNIX sites.
- T4014* Directs the output to a Tektronix 4014 terminal via the *tc*(1) filter.
- Ttek* Same as *-T4014*.
- a* Invokes the *-a* option of *troff*(1).
- y* Causes *mmt* to use the non-compacted version of the macros (see *mm*(5)). No effect for *mvt*.

These commands read the standard input when *-* is specified instead of any file names.

*Mvt* is just a link to *mmt*.

## HINT

Use the *-olist* option of *troff*(1) to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the *-e*, *-t*, and *-* options, *together* with the *-olist* option of *troff*(1) may cause a harmless “broken pipe” diagnostic if the last page of the document is not specified in *list*.

## SEE ALSO

*env*(1), *eqn*(1), *mm*(1), *tbl*(1), *tc*(1), *troff*(1), *profile*(4), *environ*(5), *mm*(5), *mv*(5).

*UNIX System Document Processing Guide*.

## DIAGNOSTICS

“*m[mv]t: no input file*” if none of the arguments is a readable file and the command is not used as a filter.

**NAME**

**net** — execute a command on the PCL network

**SYNOPSIS**

**net system [command[args]]**

**DESCRIPTION**

*Net* provides a bi-directional connection to another UNIX. The first argument is the name of the remote *system*. The second argument is a *command* to be executed. If *command* is not given, then an interactive shell (`/bin/sh -i`) on the remote system is created and an initial working directory of `/` is established. Any remaining arguments are passed to the given command as arguments.

*Net* reads the standard input, thus allowing *command* to be part of a “pipe-line” if *command* reads the standard input also.

**EXAMPLES**

Execute the *who*(1) command on system A and return the output to your terminal:

```
net A who
```

Copy a directory structure from system A to the local system:

```
cd /dir/on/localsys
net A "cd /dir/on/A; find . -print | cpio -oc" | cpio -icda
```

Copy one file from system A to the local system:

```
net A "cat /file/on/A" > /file/on/localsys
```

Send a directory structure from the local system to system A (this uses the command’s ability to read standard input):

```
find . -print | cpio -o | net A "cd /dir/on/A; cpio -id"
```

**FILES**

`/dev/pcl/?[0-7]` PCL channel interfaces for system `?`.  
`/dev/pcl/ctrl` PCL control channel.  
`/usr/adm/pcllog` activity log.

**SEE ALSO**

*cpio*(1), *find*(1), *sh*(1), *who*(1).

**DIAGNOSTICS**

*net: cannot open channel to system*

A connection can’t be made to the requested system.

*connection broken*

A non-recoverable write error occurred.

*write error*

A recoverable write error occurred. The write will be retried until it completes successfully without losing data.

*cannot fork reader process*

*Net* is unable to create a reader process and a writer process.

**WARNINGS**

A successful invocation of *net* reads at least 2 blocks of the standard input, if present, even if *command* does not use standard input. The standard input must be explicitly closed (via `<&-`) or redirected (such as from `/dev/null`) if this feature is not desired.

**BUGS**

Only the first character of a *system* name is recognized. Remaining characters are silently discarded.

The user's command environment is not carried forward to the remote system except for the effective user ID.

Executing commands that do "funny" things with your terminal (i.e., *cu*(1C), *passwd*(1), *su*(1), etc.) don't work as expected.



## NAME

**newform** — change the format of a text file

## SYNOPSIS

**newform** [-s] [-itabspec] [-otabspec] [-bn] [-en] [-pn] [-an]  
[-f] [-cchar] [-ln] [files]

## DESCRIPTION

*Newform* reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for *-s*, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like “*-e15 -l60*” will yield results different from “*-l60 -e15*”. Options are applied to all *files* on the command line.

*-itabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* may be *--*, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec*(4)). If no *tabspec* is given, *tabspec* defaults to *-8*. A *tabspec* of *-0* expects no tabs; if any are found, they are treated as *-1*.

*-otabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for *-itabspec*. If no *tabspec* is given, *tabspec* defaults to *-8*. A *tabspec* of *-0* means that no spaces will be converted to tabs on output.

*-ln* Set the effective line length to *n* characters. If *n* is not entered, *-l* defaults to 72. The default line length without the *-l* option is 80 characters. Note that tabs and backspaces are considered to be one character (use *-i* to expand tabs to spaces).

*-bn* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see *-ln*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when *-b* with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform -l1 -b7 file-name
```

The *-l1* must be used to set the effective line length shorter than any existing line in the file so that the *-b* option is activated.

*-en* Same as *-bn* except that characters are truncated from the end of the line.

*-ck* Change the prefix/append character to *k*. Default character for *k* is a space.

*-pn* Prefix *n* characters (see *-ck*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

*-an* Same as *-pn* except characters are appended to the end of a line.

**-f** Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **-o** option. If no **-o** option is specified, the line which is printed will contain the default specification of **-8**.

**-s** Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a \* and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

```
newform -s -i -l -a -e file-name
```

## DIAGNOSTICS

All diagnostics are fatal.

|                                    |                                                                                                                                      |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>usage: ...</i>                  | <i>Newform</i> was called with a bad option.                                                                                         |
| <i>not -s format</i>               | There was no tab on one line.                                                                                                        |
| <i>can't open file</i>             | Self explanatory.                                                                                                                    |
| <i>internal line too long</i>      | A line exceeds 512 characters after being expanded in the internal work buffer.                                                      |
| <i>tabspec in error</i>            | A tab specification is incorrectly formatted, or specified tab stops are not ascending.                                              |
| <i>tabspec indirection illegal</i> | A <i>tabspec</i> read from a file (or standard input) may not contain a <i>tabspec</i> referencing another file (or standard input). |

## EXIT CODES

0 — normal execution  
1 — for any error

## SEE ALSO

csplit(1), tabs(1), fspec(4).

## BUGS

*Newform* normally only keeps track of physical characters; however, for the **-i** and **-o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

*Newform* will not prompt the user if a *tabspec* is to be read from the standard input (by use of **-i--** or **-o--**).

If the **-f** option is used, and the last **-o** option specified was **-o--**, and was preceded by either a **-o--** or a **-i--**, the tab specification format line will be incorrect.

**NAME**

**newgrp** — log in to a new group

**SYNOPSIS**

**newgrp** [-] [ group ]

**DESCRIPTION**

*Newgrp* changes the group identification of its caller, analogously to *login*(1). The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

*Newgrp* without an argument changes the group identification to the group in the password file; in effect it changes the group identification back to the caller's original group.

An initial - flag causes the environment to be changed to the one that would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user himself 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.

When most users log in, they are members of the group named *other*.

**FILES**

*/etc/group*  
*/etc/passwd*

**SEE ALSO**

*login*(1), *group*(4).

**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**

*news* — print news items

**SYNOPSIS**

*news* [ *-a* ] [ *-n* ] [ *-s* ] [ *items* ]

**DESCRIPTION**

*News* is used to keep the user informed of current events. By convention, these events are described by files in the directory */usr/news*.

When invoked without arguments, *news* prints the contents of all current files in */usr/news*, most recent first, with each preceded by an appropriate header. *News* stores the "currency" time as the modification date of a file named *.news\_time* in the user's home directory (the identity of this directory is determined by the environment variable *\$HOME*); only files more recent than this currency time are considered "current."

The *-a* option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

The *-n* option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

The *-s* option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's *.profile* file, or in the system's */etc/profile*.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

**FILES**

*/etc/profile*  
*/usr/news/\**  
*\$HOME/.news\_time*

**SEE ALSO**

*profile(4)*, *environ(5)*.

**NAME**

*nice* — run a command at low priority

**SYNOPSIS**

*nice* [ *-increment* ] *command* [ *arguments* ]

**DESCRIPTION**

*Nice* executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., *--10*.

**SEE ALSO**

*nohup*(1), *nice*(2).

**DIAGNOSTICS**

*Nice* returns the exit status of the subject command.

**BUGS**

An *increment* larger than 19 is equivalent to 19.

## NAME

**nl** — line numbering filter

## SYNOPSIS

**nl** [**-h**type] [**-b**type] [**-f**type] [**-v**start#] [**-i**incr] [**-p**] [**-l**num]  
 [**-s**sep] [**-w**width] [**-n**format] [**-d**delim] file

## DESCRIPTION

*Nl* reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

*Nl* views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g. no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

| <i>Line contents</i> | <i>Start of</i> |
|----------------------|-----------------|
| \\: \\:              | header          |
| \\:                  | body            |
| \\:                  | footer          |

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

- b**type Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **p**string, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is **t** (text lines numbered).
- h**type Same as **-b**type except for header. Default *type* for logical page header is **n** (no lines numbered).
- f**type Same as **-b**type except for footer. Default for logical page footer is **n** (no lines numbered).
- p** Do not restart numbering at logical page delimiters.
- v**start# *Start#* is the initial value used to number logical page lines. Default is **1**.
- i**incr *Incr* is the increment value used to number logical page lines. Default is **1**.
- s**sep *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.
- w**width *Width* is the number of characters to be used for the line number. Default *width* is **6**.
- n**format *Format* is the line numbering format. Recognized values are: **l**m, left justified, leading zeroes suppressed; **r**n, right justified, leading zeroes suppressed; **r**z, right justified, leading zeroes kept. Default *format* is **r**n (right justified).

- 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). 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!+ file1 file2
```

will number files 1 and 2 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

**SEE ALSO**

pr(1).

**NAME**

**nm** — print name list of common object file

**SYNOPSIS**

**nm** [-o] [-x] [-h] [-v] [-n] [-e] [-f] [-u] [-V] file-names

**DESCRIPTION**

The *nm* command displays the symbol table of each common object file *file-name*. *File-name* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>    | The name of the symbol.                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Value</b>   | Its value expressed as an offset or an address depending on its storage class.                                                                                                                                                                                                                                                                                                                                                    |
| <b>Class</b>   | Its storage class.                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Type</b>    | Its type and derived type. If the symbol is an instance of a structure or of a union then the structure or union tag will be given following the type (e.g. struct-tag). If the symbol is an array, then the array dimensions will be given following the type (eg., <b>char[n][m]</b> ). Note that the object file must have been compiled with the <b>-g</b> option of the <i>cc(1)</i> command for this information to appear. |
| <b>Size</b>    | Its size in bytes, if available. Note that the object file must have been compiled with the <b>-g</b> option of the <i>cc(1)</i> command for this information to appear.                                                                                                                                                                                                                                                          |
| <b>Line</b>    | The source line number at which it is defined, if available. Note that the object file must have been compiled with the <b>-g</b> option of the <i>cc(1)</i> command for this information to appear.                                                                                                                                                                                                                              |
| <b>Section</b> | For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).                                                                                                                                                                                                                                                                                                                 |

The output of *nm* may be controlled using the following flags:

- o** A symbol's value and size will be printed in octal instead of decimal.
- x** A symbol's value and size will be printed in hexadecimal instead of decimal.
- h** The output header data is not displayed.
- v** External symbols will be sorted by value before they are printed.
- n** External symbols will be sorted by name before they are printed.
- e** Only static and external symbols are printed.
- f** Full output is produced. Redundant symbols (.text, .data and .bss), normally suppressed, are printed.
- u** Only undefined symbols are printed.
- V** Version of *nm* command executing is displayed on stderr output.

Flags may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **nm name -e -v** and **nm -ve name** print the static and external symbols in *name*, with external symbols sorted by value.

**FILES**

*/usr/tmp/nm?????*



**SEE ALSO**

as(1), cc(1), ld(1), a.out(4), ar(4).

**DIAGNOSTICS**

“nm: name: cannot open”

if *name* cannot be read.

“nm: name: bad magic”

if *name* is not an appropriate common object file.

“nm: name: no symbols”

if the symbols have been stripped from *name*.

**NAME**

**nm** — print name list

**SYNOPSIS**

**nm** [ **-gnoprsu** ] [ **file ...** ]

**DESCRIPTION**

*Nm* prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in **a.out** are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

- g** Print only global (external) symbols.
- n** Sort numerically rather than alphabetically.
- o** Prefix file or archive element name to each output line rather than only once. This option can be used to make piping to *grep*(1) more meaningful.
- p** Don't 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 highest value). This difference is the value printed. This flag turns on **-g** and **-n** and turns off **-u** and **-p**.
- u** Print only undefined symbols.

**SEE ALSO**

**ar**(1), **a.out**(4), **ar**(4).

**NAME**

**nohup** — run a command immune to hangups and quits

**SYNOPSIS**

**nohup** *command* [ *arguments* ]

**DESCRIPTION**

*Nohup* executes *command* with hangups and quits ignored. If output is not re-directed by the user, it will be sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **\$HOME/nohup.out**.

**SEE ALSO**

**nice(1)**, **signal(2)**.

## NAME

nroff — format text

## SYNOPSIS

nroff [ options ] [ files ]

## DESCRIPTION

*Nroff* formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers. Its capabilities are described in the *NROFF/TROFF User's Manual* cited below.

An argument consisting of a minus (-) is taken to be a file name corresponding to the standard input. The *options*, which may appear in any order, but must appear before the *files*, are:

- olist Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. 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. (See *BUGS* below.)
- nN Number first generated page *N*.
- sN Stop every *N* pages. *Nroff* will halt *after* every *N* pages (default *N=1*) to allow paper loading or changing, and will resume upon receipt of a line-feed or new-line (new-lines do not work in pipelines, e.g., with *mm*(1)). This option does not work if the output of *nroff* is piped through *col*(1). When *nroff* halts between pages, an ASCII BEL is sent to the terminal.
- raN Set register *a* (which must have a one-character name) to *N*.
- i Read standard input after *files* are exhausted.
- q Invoke the simultaneous input-output mode of the *.rd* request.
- z Print only messages generated by *.tm* (terminal message) requests.
- mname Prepend to the input *files* the non-compacted (ASCII text) macro file */usr/lib/tmac/tmac.name*.
- cname Prepend to the input *files* the compacted macro files */usr/lib/macros/cmp.[nt].[dt].name* and */usr/lib/macros/ucmp.[nt].name*.
- kname Compact the macros used in this invocation of *nroff*, placing the output in files *[dt].name* in the current directory (see the May 1979 Addendum to the *NROFF/TROFF User's Manual* for details of compacting macro files).
- Tname Prepare output for specified terminal. Known *names* are **37** for the (default) TELETYPE® Model 37 terminal, **tm300** for the GE TerminiNet 300 (or any terminal without half-line capability), **300s** for the DASI 300s, **300** for the DASI 300, **450** for the DASI 450, **lp** for a (generic) ASCII line printer, **382** for the DTC-382, **4000A** for the Trendata 4000A, **832** for the Anderson Jacobson 832, **X** for a (generic) EBCDIC printer, and **2631** for the Hewlett Packard 2631 line printer.
- e Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.
- 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.
- un Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

## FILES

|                      |                                          |
|----------------------|------------------------------------------|
| /usr/lib/suftab      | suffix hyphenation tables                |
| /tmp/ta\$#           | temporary file                           |
| /usr/lib/tmac/tmac.* | standard macro files and pointers        |
| /usr/lib/macros/*    | standard macro files                     |
| /usr/lib/term/*      | terminal driving tables for <i>nroff</i> |

## SEE ALSO

*NROFF/TROFF User's Manual*

*A TROFF Tutorial*

col(1), cw(1), eqn(1), greek(1), mm(1), tbl(1), troff(1), mm(5).

## BUGS

*Nroff* believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *nroff* generates may be off by one day from your idea of what the date is.

When *nroff* is used with the *-olist* option inside a pipeline (e.g., with one or more of *cw(1)*, *eqn(1)*, and *tbl(1)*), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

**NAME**

nscstat — query the operation status of the NSC network

**SYNOPSIS**

nscstat [ netname ... ] [-ludqrbpa] [-n names ]

**DESCRIPTION**

*Nscstat*, without arguments, gives a short operational status report of the NSC network from the viewpoint of the local node. This includes the status of the NSC network and the total number of files queued for transmission. A list of network names may be specified. If no network names are given, the options specified are performed for all known networks. *Nscstat* recognizes the following arguments:

- l Output a long listing. This option indicates the status of the printed node (on-line or off-line), the total number of files queued waiting transmission to this node, and the time when the first job was queued for transmission.
- p Report the last time the system received a *poke* from remote systems.
- r Report the last time the system received a request to transfer from remote systems.
- b Report the last time the current system had to notify remote systems that it was too busy to handle its request.
- u List the status of all nodes that are on-line (up).
- d List the status of all nodes that are off-line (down).
- q List the status of all nodes that have files queued for transmission.
- a List the status of all nodes configured on the network.
- n names  
*Name* specifies that status is requested for this node only. If more than one network is specified, this option is disabled.

Each of the above arguments may be used singly or together with several others. When used together, the output is the intersection of the sets of nodes matching each option. If a node name list is specified, status for that node will only be reported if it is in the intersection set of the specified options.

**EXAMPLE**

To get a long listing of all nodes that are currently off-line and have files queued for transmission:

```
nscstat -ldq
```

**FILES**

|                         |                                                          |
|-------------------------|----------------------------------------------------------|
| /usr/nsc/rvchan         | list of nodes currently configured on the network        |
| /usr/nsc/cons/*         | nodes that are considered on-line                        |
| /usr/nsc/jobs/C*        | jobs queued for transmission                             |
| /usr/nsc/cons/on-line/* | whether the NSC network is active or not on this network |

**BUGS**

*Nscstat* tries to interpret the specified options intelligently. If none of the options specified apply to any of the specified nodes, no detailed status will be reported.

**NAME**

nsctorje — re-route jobs from the NSC network to RJE

**SYNOPSIS**

nsctorje [-d names]

**DESCRIPTION**

*Nsctorje* will resubmit jobs queued on the NSC local network (via *nusend*(1C) across the RJE link (if it exists). *Nsctorje* submits a *nusend*(1C) command to re-route each queued job. By default, jobs will be re-routed if either the remote host is marked down locally or if the NSC network on the local host is inactive. *Nsctorje* recognizes the following options:

-d names re-route all jobs queued only to the remote machine *name*.

**SEE ALSO**

*nusend*(1C).  
*nscmon*(1M), *rje*(8) in the *UNIX System Administrator's Manual*.

**FILES**

|                 |                                                               |
|-----------------|---------------------------------------------------------------|
| /usr/nsc/NORJE  | file indicating that no RJE connection exists on this machine |
| /usr/nsc/rvchan | <i>nusend</i> (1C) network configuration file                 |
| /usr/asp/udest  | nodes accessible through RJE                                  |

**BUGS**

Any file larger than 190,000 bytes will not be re-routed across the RJE link. It will remain queued on the NSC network until the remote node becomes available.

**NAME**

**nusend** — send files to another UNIX on the NSC network

**SYNOPSIS**

**nusend** **-d** *dest* [**-n** *netname*] [**-a** *acct*] [**-m**] [**-e**] [**-s**] [**-c**] [**-x**] [**-u** *destuser*] [[**-f** *destfile*] *srcfile*] [**-!cmd** [*cmdfile*]] ...

**DESCRIPTION**

*Nusend* sends copies of the named files or command to another UNIX system via the NSC network. If the file name **-** is given, the standard input is read at that point.

**-d** *dest* Destination. *Dest* can be any one of the UNIX systems on the NSC local network. See */usr/nsc/rvchan* for an up-to-date list of valid NSC destinations.

**-n** *netname*

Network name. *Netname* can be any one of the networks known to the local system (see *nscmon(1M)* for the definition of a network. This option is only needed when sending to your own system. See */usr/nsc/nets* for the up-to-date list of valid networks).

**-a** *acct* Use *acct* as the account number for the job. By default, the account number is read from the password file.

**-s** Silent. Suppress the one-line message which contains the submitted job name.

**-c** Copy. Make a copy of the file. Default is to set up a pointer to the file in the user's directory. If any changes are made to the file before transmission, the changes will be sent to the destination unless the **-c** option is used.

**-x** Generate checksums on all data transmissions.

Mail will normally be sent to the receiving login(s) to report the receipt of the file(s). Mail will be sent to both sending and receiving logins if there were errors in transmission. The default may be overridden with the following switches:

**-m** Report by *mail(1)* when the file transfer is complete. The mail is sent from the remote system via *nusend*.

**-e** Report by *mail(1)* only when an error occurred during the transfer. No other mail will be sent.

Normally, the login name under which the new file will appear on the destination system is the same as the login name of the person who issues the command.

The following options, each as a separate argument, may be interspersed with file name arguments:

**-u** Use the next argument as the destination user's login name for all succeeding files.

**-f** Use the next argument as the destination file name for the succeeding file. *Srcfile* must be specified. The destination path name is assumed to be relative to the destination login directory if there is no leading **/**. In either case, the target directory must be mode *777*, or if the file already exists, the file must be writable by others. By default, files are delivered to directory *rje* under the destination login directory. *Rje* must have been previously created in mode *777* for everything to work. The name of the destination file is ordinarily the same as the last component of the



original file. When the standard input is sent, the destination file name is normally taken to be `pipe.end`. If `-` is used, the standard input is taken.

`!cmd` *Cmd* is sent to the remote machine for execution. A file name or `-` can be used as standard input to the command. If no file is specified, `/dev/null` is used.

**EXAMPLES**

Assuming `XXAAA`, `XXBBB` and `XXCCC` are machines on the NSC network, then:

To send files *file1*, *file2*, and *file3* to `XXAAA` (assuming the source and destination logins are the same):

```
nusend -d XXAAA file1 file2 file3
```

To send file `cprog.c` to login name `dave` on `XXBBB` and to get confirmation mail returned:

```
nusend -d XXBBB -m -u dave cprog.c
```

To send file `myfile` to `XXCCC` and rename it to `yourfile` (assuming the source and destination logins are the same):

```
nusend -d XXCCC -f yourfile myfile
```

To send file `a.out` from `XXAAA` to login name `debbie` on `XXBBB` via remote execution:

```
nusend -d XXAAA .RS -!`nusend -d XXBBB -u debbie`  
`logdir debbie`/a.out`
```

**FILES**

|                                  |                             |
|----------------------------------|-----------------------------|
| <code>/etc/passwd</code>         | account number for NSC job  |
| <code>/usr/nsc/jobs/C*</code>    | job queue area              |
| <code>/usr/nsc/rvchan</code>     | table of known destinations |
| <code>/usr/nsc/nets</code>       | table of known networks     |
| <code>/usr/nsc/log/nusend</code> | usage log                   |

**SEE ALSO**

`mail(1)`, `nscstat(1C)`.



**NAME**

od - octal dump

**SYNOPSIS**

od [ -bcdosx ] [ file ] [ [ + ]offset[ . ][ b ] ]

**DESCRIPTION**

*Od* dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, -o is default. The meanings of the format options are:

- b Interpret bytes in octal.
- c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.
- d Interpret words in unsigned decimal.
- o Interpret words in octal.
- s Interpret 16-bit words in signed decimal.
- x Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If . is appended, the offset is interpreted in decimal. If b is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by +.

Dumping continues until end-of-file.

**SEE ALSO**

dump(1).

**NAME**

*pack*, *pcat*, *unpack* — compress and expand files

**SYNOPSIS**

**pack** [ - ] *name* ...

**pcat** *name* ...

**unpack** *name* ...

**DESCRIPTION**

*Pack* attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

*Pack* uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the - argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of - in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. 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 character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, 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, the packed versions being about 90% of the original size.

*Pack* returns a value that is the number of files that it failed to compress.

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 file name must contain no more than 12 characters to allow space for the appended *.z* extension. Directories cannot be compressed.

*Pcat* does for packed files what *cat(1)* does for ordinary files. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name.z* use:

*pcat name.z*

or just:

*pcat name*

To make an unpacked copy, say *nnn*, of a packed file named *name.z* (without destroying *name.z*) use the command:

*pcat name > nnn*

*Pcat* returns the number of files it was unable to unpack. Failure may occur if:

- the file name (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 *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in *.z*). If this file appears to be a packed file, 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.

*Unpack* returns a value that is the number of files it was unable to unpack. 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;
- if the unpacked file cannot be created.

**NAME**

passwd — change login password

**SYNOPSIS**

passwd name

**DESCRIPTION**

This command changes (or installs) a password associated with the login *name*.

The program prompts for the old password (if any) and then for the new one (twice). The caller must supply these. New passwords should be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monospace. Only the first eight characters of the password are significant.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password. Only the super-user can create a null password.

The password file is not changed if the new password is the same as the old password, or if the password has not “aged” sufficiently; see *passwd(4)*.

**FILES**

/etc/passwd

**SEE ALSO**

login(1), crypt(3C), passwd(4).

**NAME**

**paste** — merge same lines of several files or subsequent lines of one file

**SYNOPSIS**

```
paste file1 file2 ...
paste -d list file1 file2 ...
paste -s [-d list] file1 file2 ...
```

**DESCRIPTION**

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* subsumes 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*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if *-* is used in place of a file name.

The meanings of the options are:

- d** Without this option, the new-line 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 (see below).
- list* One or more characters immediately following *-d* replace the default *tab* as the line concatenation character. The *list* is used circularly, i. e. when exhausted, it is reused. In parallel merging (i. e. no *-s* option), the lines from the last file are always terminated with a new-line character, not from the *list*. The *list* may contain the special escape sequences: *\n* (new-line), *\t* (*tab*), *\* (backslash), and *\0* (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g. to get one backslash, use *-d"\\\\"*).
- s** Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with *-d* option. Regardless of the *list*, the very last character of the file is forced to be a new-line.
- May be used in place of any file name, to read a line from the standard input. (There is no prompting).

**EXAMPLES**

```
ls | paste -d" " -           list directory in one column
ls | paste - - - -          list directory in four columns
paste -s -d"\t\n" file      combine pairs of lines into lines
```

**SEE ALSO**

*grep*(1), *cut*(1),  
*pr*(1): *pr -t -m...* works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

**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

`pr` — print files

## SYNOPSIS

`pr` [ options ] [ files ]

## DESCRIPTION

`Pr` prints the named files on the standard output. If *file* is `-`, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the `-s` option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until `pr` has completed printing.

The below *options* may appear singly or be combined in any order:

- `+k` Begin printing with page *k* (default is 1).
- `-k` Produce *k*-column output (default is 1). The options `-e` and `-i` are assumed for multi-column output.
- `-a` Print multi-column output across the page.
- `-m` Merge and print all files simultaneously, one per column (overrides the `-k`, and `-a` options).
- `-d` Double-space the output.
- `-eck` Expand *input* tabs 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 spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).
- `-ick` In *output*, replace white space wherever possible by inserting tabs 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 character (default for *c* is the tab character).
- `-nck` 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).
- `-wk` Set the width of a line to *k* character positions (default is 72 for equal-width multi-column output, no limit otherwise).
- `-ok` Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.
- `-lk` Set the length of a page to *k* lines (default is 66).
- `-h` Use the next argument as the header to be printed instead of the file name.
- `-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 carriage return).

- f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.
- r Print no diagnostic reports on failure to open files.
- t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.
- sc Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

**EXAMPLES**

Print `file1` and `file2` as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Write `file1` on `file2`, expanding tabs to columns 10, 19, 28, 37, ...:

```
pr -e9 -t <file1 >file2
```

**FILES**

`/dev/tty*` to suspend messages

**SEE ALSO**

`cat(1)`.



**NAME**

prof — display profile data

**SYNOPSIS**

prof [-tcn] [-ox] [-g] [-z] [-h] [-s] [-m mdata] [prog]

**DESCRIPTION**

*Prof* interprets the profile file produced by the *monitor*(3C) function. The symbol table in the object file *prog* (*a.out* by default) is read and correlated with the profile file (*mon.out* by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options **t**, **c**, **a**, and **n** determine the type of sorting of the output lines:

- t Sort by decreasing percentage of total time (default).
- c Sort by decreasing number of calls.
- a Sort by increasing symbol address.
- n Sort lexically by symbol name.

The mutually exclusive options **o** and **x** specify the printing of the address of each symbol monitored:

- o Print each symbol address (in octal) along with the symbol name.
- x Print each symbol address (in hexadecimal) along with the symbol name.

The following options may be used in any combination:

- g Include non-global symbols (static functions).
- z Include all symbols in the profile range (see *monitor*(3C)), even if associated with zero number of calls and zero time.
- h Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)
- s Print a summary of several of the monitoring parameters and statistics on the standard error output.
- m mdata  
Use file *mdata* instead of *mon.out* for profiling data.

For the number of calls to a function to be tallied, the **-p** option of *cc*(1) must have been given when the file containing the function was compiled. This option to the *cc* command also arranges for the object file to include a special profiling start-up function that calls *monitor*(3C) at the beginning and end of execution. It is the call to *monitor* at the end of execution that causes the *mon.out* file to be written. Thus, only programs that call *exit*(2) or return from *main* will cause the *mon.out* file to be produced.

**FILES**

*mon.out* for profile  
*a.out* for namelist

**SEE ALSO**

*cc*(1), *nm*(1), *exit*(2), *profil*(2), *monitor*(3C).

**BUGS**

There is a limit of 300 functions that may have call counters established during program execution. If this limit is exceeded, other data will be overwritten and the `mon.out` file will be corrupted. The number of call counters used will be reported automatically by the *prof* command whenever the number exceeds 250.

**NAME**

*prs* — print an SCCS file

**SYNOPSIS**

*prs* [-d[*dataspec*]] [-r[SID]] [-e] [-l] [-a] files

**DESCRIPTION**

*Prs* prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(4)) in a user supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.), and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

- d[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.
- r[SID] Used to specify the SCCS *ID*entification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.
- e Requests information for all deltas created *earlier* than and including the delta designated via the -r *keyletter*.
- l Requests information for all deltas created *later* than and including the delta designated via the -r *keyletter*.
- a Requests printing of information for both removed, i.e., delta type = *R*, (see *rmDEL*(1)) and existing, i.e., delta type = *D*, deltas. If the -a *keyletter* is not specified, information for existing deltas only is provided.

**DATA KEYWORDS**

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(4)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by \t and carriage return/new-line is specified by \n.

TABLE 1. SCCS Files Data Keywords

| Keyword | Data Item                               | File Section | Value             | Format |
|---------|-----------------------------------------|--------------|-------------------|--------|
| :DT:    | Delta information                       | Delta Table  | See below*        | S      |
| :DL:    | Delta line statistics                   | "            | :Li/:Ld/:Lu:      | S      |
| :Li:    | Lines inserted by Delta                 | "            | nnnnn             | S      |
| :Ld:    | Lines deleted by Delta                  | "            | nnnnn             | S      |
| :Lu:    | Lines unchanged by Delta                | "            | nnnnn             | S      |
| :DT:    | Delta type                              | "            | D or R            | S      |
| :I:     | SCCS ID string (SID)                    | "            | :R::L::B::S:      | S      |
| :R:     | Release number                          | "            | nnnn              | S      |
| :L:     | Level number                            | "            | nnnn              | S      |
| :B:     | Branch number                           | "            | nnnn              | S      |
| :S:     | Sequence number                         | "            | nnnn              | S      |
| :D:     | Date Delta created                      | "            | :Dy/:Dm/:Dd:      | S      |
| :Dy:    | Year Delta created                      | "            | nn                | S      |
| :Dm:    | Month Delta created                     | "            | nn                | S      |
| :Dd:    | Day Delta created                       | "            | nn                | S      |
| :T:     | Time Delta created                      | "            | :Th::Tm::Ts:      | S      |
| :Th:    | Hour Delta created                      | "            | nn                | S      |
| :Tm:    | Minutes Delta created                   | "            | nn                | S      |
| :Ts:    | Seconds Delta created                   | "            | nn                | S      |
| :P:     | Programmer who created Delta            | "            | logname           | S      |
| :DS:    | Delta sequence number                   | "            | nnnn              | S      |
| :DP:    | Predecessor Delta seq-no.               | "            | nnnn              | S      |
| :DI:    | Seq-no. of deltas incl., excl., ignored | "            | :Dn/:Dx/:Dg:      | S      |
| :Dn:    | Deltas included (seq #)                 | "            | :DS: :DS: ...     | S      |
| :Dx:    | Deltas excluded (seq #)                 | "            | :DS: :DS: ...     | S      |
| :Dg:    | Deltas ignored (seq #)                  | "            | :DS: :DS: ...     | S      |
| :MR:    | MR numbers for delta                    | "            | text              | M      |
| :C:     | Comments for delta                      | "            | text              | M      |
| :UN:    | User names                              | User Names   | text              | M      |
| :FL:    | Flag list                               | Flags        | text              | M      |
| :Y:     | Module type flag                        | "            | text              | S      |
| :MF:    | MR validation flag                      | "            | yes or no         | S      |
| :MP:    | MR validation pgm name                  | "            | text              | S      |
| :KF:    | Keyword error/warning flag              | "            | yes or no         | S      |
| :BF:    | Branch flag                             | "            | yes or no         | S      |
| :J:     | Joint edit flag                         | "            | yes or no         | S      |
| :LK:    | Locked releases                         | "            | :R: ...           | S      |
| :Q:     | User defined keyword                    | "            | text              | S      |
| :M:     | Module name                             | "            | text              | S      |
| :FB:    | Floor boundary                          | "            | :R:               | S      |
| :CB:    | Ceiling boundary                        | "            | :R:               | S      |
| :Ds:    | Default SID                             | "            | :I:               | S      |
| :ND:    | Null delta flag                         | "            | yes or no         | S      |
| :FD:    | File descriptive text                   | Comments     | text              | M      |
| :BD:    | Body                                    | Body         | text              | M      |
| :GB:    | Gotten body                             | "            | text              | M      |
| :W:     | A form of <i>what</i> (1) string        | N/A          | :Z::M:\t:I:       | S      |
| :A:     | A form of <i>what</i> (1) string        | N/A          | :Z::Y: :M: :I::Z: | S      |
| :Z:     | <i>what</i> (1) string delimiter        | N/A          | @( # )            | S      |
| :F:     | SCCS file name                          | N/A          | text              | S      |
| :PN:    | SCCS file path name                     | N/A          | text              | S      |

\* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

1

**EXAMPLES**

prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

Users and/or user IDs for s.file are:  
 xyz  
 131  
 abc

prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file

may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a *special case*:

prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000  
 MRs:  
 bl78-12345  
 bl79-54321  
 COMMENTS:  
 this is the comment line for s.file initial delta

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the -a keyletter.

**FILES**

/tmp/pr????

**SEE ALSO**

admin(1), delta(1), get(1), help(1), sccsfile(4).  
*Source Code Control System User's Guide* in the *UNIX System User's Guide*.

**DIAGNOSTICS**

Use *help*(1) for explanations.

1

## NAME

ps — report process status

## SYNOPSIS

ps [ options ]

## DESCRIPTION

*Ps* prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. Otherwise, the information that is displayed is controlled by the following *options*:

- e           Print information about all processes.
- d           Print information about all processes, except process group leaders.
- a           Print information about all processes, except process group leaders and processes not associated with a terminal.
- f           Generate a *full* listing. (Normally, a short listing containing only process ID, terminal (“tty”) identifier, cumulative execution time, and the command name is printed.) See below for meaning of columns in a full listing.
- l           Generate a *long* listing. See below.
- c *corefile* Use the file *corefile* in place of /dev/mem.
- s *swapdev* Use the file *swapdev* in place of /dev/swap. This is useful when examining a *corefile*; a *swapdev* of /dev/null will cause the user block to be zeroed out.
- n *namelist* The argument will be taken as the name of an alternate *namelist* (/unix is the default).
- t *tlist*     Restrict listing to data about the processes associated with the terminals given in *tlist*, where *tlist* can be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.
- p *plist*    Restrict listing to data about processes whose process ID numbers are given in *plist*, where *plist* is in the same format as *tlist*.
- u *ulist*     Restrict listing to data about processes whose user ID numbers or login names are given in *ulist*, where *ulist* is in the same format as *tlist*. In the listing, the numerical user ID will be printed unless the —f option is used, in which case the login name will be printed.
- g *glist*     Restrict listing to data about processes whose process groups are given in *glist*, where *glist* is a list of process group leaders and is in the same format as *tlist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters f and l indicate the option (*full* or *long*) that causes the corresponding heading to appear; all means that the heading always appears. Note that these two options only determine what information is provided for a process; they do *not* determine which processes will be listed.

- F           (l)   Flags (octal and additive) associated with the process:
- 01    in core;
  - 02    system process;
  - 04    locked in core (e.g., for physical I/O);
  - 10    being swapped;
  - 20    being traced by another process;

|       |       |                                                                                                                                  |                       |
|-------|-------|----------------------------------------------------------------------------------------------------------------------------------|-----------------------|
|       |       | 40                                                                                                                               | another tracing flag. |
| S     | (l)   | The state of the process:                                                                                                        |                       |
|       |       | 0                                                                                                                                | non-existent;         |
|       |       | S                                                                                                                                | sleeping;             |
|       |       | W                                                                                                                                | waiting;              |
|       |       | R                                                                                                                                | running;              |
|       |       | I                                                                                                                                | intermediate;         |
|       |       | Z                                                                                                                                | terminated;           |
|       |       | T                                                                                                                                | stopped;              |
|       |       | X                                                                                                                                | growing.              |
| UID   | (f,l) | The user ID number of the process owner; the login name is printed under the <code>-f</code> option.                             |                       |
| PID   | (all) | The process ID of the process; it is possible to kill a process if you know this datum.                                          |                       |
| PPID  | (f,l) | The process ID of the parent process.                                                                                            |                       |
| C     | (f,l) | Processor utilization for scheduling.                                                                                            |                       |
| STIME | (f)   | Starting time of the process.                                                                                                    |                       |
| PRI   | (l)   | The priority of the process; higher numbers mean lower priority.                                                                 |                       |
| NI    | (l)   | Nice value; used in priority computation.                                                                                        |                       |
| ADDR  | (l)   | The memory address of the process (a pointer to the segment table array on the 3B20S), if resident; otherwise, the disk address. |                       |
| SZ    | (l)   | The size in blocks of the core image of the process.                                                                             |                       |
| WCHAN | (l)   | The event for which the process is waiting or sleeping; if blank, the process is running.                                        |                       |
| TTY   | (all) | The controlling terminal for the process.                                                                                        |                       |
| TIME  | (all) | The cumulative execution time for the process.                                                                                   |                       |
| CMD   | (all) | The command name; the full command name and its arguments are printed under the <code>-f</code> option.                          |                       |

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked `<defunct>`.

Under the `-f` option, `ps` tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the `-f` option, is printed in square brackets.

#### FILES

|                           |                                          |
|---------------------------|------------------------------------------|
| <code>/unix</code>        | system namelist.                         |
| <code>/dev/mem</code>     | memory.                                  |
| <code>/dev/swap</code>    | the default swap device.                 |
| <code>/etc/passwd</code>  | supplies UID information.                |
| <code>/etc/ps_data</code> | internal data structure.                 |
| <code>/dev</code>         | searched to find terminal ("tty") names. |

#### SEE ALSO

`kill(1)`, `nice(1)`.

#### BUGS

Things can change while `ps` is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

## NAME

`ptx` — permuted index

## SYNOPSIS

`ptx` [ options ] [ input [ output ] ]

## DESCRIPTION

*Ptx* generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). It 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 each line. *Ptx* output is in the form:

`.xx "tail" "before keyword" "keyword and after" "head"`

where `.xx` is assumed to be an *nroff* or *troff*(1) macro provided by the user, or provided by the *mptx*(5) macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following *options* can be applied:

- `-f` Fold upper and lower case letters for sorting.
- `-t` Prepare the output for the phototypesetter.
- `-w n` Use the next argument, *n*, as the length of the output line. The default line length is 72 characters for *nroff* and 100 for *troff*.
- `-g n` Use the next argument, *n*, as the number of characters that *ptx* will reserve in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.
- `-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. Tab, new-line, and space characters are *always* used as break characters.
- `-r` Take any leading non-blank 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.

The index for this manual was generated using *ptx*.

## FILES

`/bin/sort`  
`/usr/lib/eign`  
`/usr/lib/tmac/tmac.ptx`

## SEE ALSO

*nroff*(1), *troff*(1), *mm*(5), *mptx*(5).

## BUGS

Line length counts do not account for overstriking or proportional spacing. Lines that contain tildes (~) are botched, because *ptx* uses that character internally.



**NAME**

`pwd` — working directory name

**SYNOPSIS**

`pwd`

**DESCRIPTION**

*Pwd* prints the path name of the working (current) directory.

**SEE ALSO**

`cd(1)`.

**DIAGNOSTICS**

“Cannot open ..” and “Read error in ..” indicate possible file system trouble and should be referred to a UNIX programming counselor.

**NAME**

ratfor — rational Fortran dialect

**SYNOPSIS**

ratfor [ options ] [ files ]

**DESCRIPTION**

*Ratfor* converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:

```
{ statement; statement; statement }
```

decision-making:

```
if (condition) statement [ else statement ]
```

```
switch (integer value) {
    case integer:  statement
```

```
    ...
```

```
    [ default: ]  statement
```

```
}
```

loops:

```
while (condition) statement
```

```
for (expression; condition; expression) statement
```

```
do limits statement
```

```
repeat statement [ until (condition) ]
```

```
break
```

```
next
```

and some syntactic sugar to make programs easier to read and write:

free form input:

```
multiple statements/line; automatic continuation
```

comments:

```
# this is a comment.
```

translation of relationals:

```
>, >=, etc., become .GT., .GE., etc.
```

return expression to caller from function:

```
return (expression)
```

define:

```
define name replacement
```

include:

```
include file
```

The option **-h** causes quoted strings to be turned into **27H** constructs. The **-C** option copies comments to the output and attempts to format it neatly. Normally, continuation lines are marked with a **&** in column 1; the option **-6x** makes the continuation character **x** and places it in column 6.

*Ratfor* is best used with *f77(1)*.

**SEE ALSO**

*cfl(1)*, *f77(1)*.

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

**NAME**

`regcmp` — regular expression compile

**SYNOPSIS**

`regcmp [ - ] files`

**DESCRIPTION**

*Regcmp*, in most cases, precludes the need for calling *regcmp(3X)* from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file.i*. If the `-` option is used, the output will be placed in *file.c*. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as extern char vectors. *File.i* files may thus be *included* into C programs, or *file.c* files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex(abc,line)* will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

**EXAMPLES**

```
name "([A-Za-z][A-Za-z0-9_]*)$0"
telno "\({0,1}([2-9][01][1-9])$0\) {0,1} *"
      "([2-9][0-9]{2})$1 [ -]{0,1}"
      "([0-9]{4})$2"
```

In the C program that uses the *regcmp* output,  
`regex(telno, line, area, exch, rest)`  
 will apply the regular expression named *telno* to *line*.

**SEE ALSO**

`regcmp(3X)`.

**NAME**

`rjestat` — RJE status report and interactive status console

**SYNOPSIS**

`rjestat` [ *host* ]... [ *-shost* ] [ *-chost cmd* ]...

**DESCRIPTION**

*Rjestat* provides a method of determining the status of an RJE link and of simulating an IBM remote console (with UNIX features added). When invoked with no arguments, *rjestat* reports the current status of all the RJE links connected to the UNIX system. The options are:

*host*            Print the status of the line to *host*. *Host* is the pseudonym for a particular IBM system. It can be any name that corresponds to one in the first column of the RJE configuration file.

*-shost*          After all the arguments have been processed, start an interactive status console to *host*.

*-chost cmd*

Interpret *cmd* as if it were entered in status console mode to *host*. See below for the proper format of *cmd*.

In status console mode, *rjestat* prompts with the host pseudonym followed by `:` whenever it is ready to accept a command. Commands are terminated with a new-line. A line that begins with `!` is sent to the UNIX shell for execution. A line that begins with the letter `q` terminates *rjestat*. All other input lines are assumed to have the form:

*ibmcmd* [ *redirect* ]

*Ibmcmd* is any IBM JES or HASP command. Only the super-user or *rje* login can send commands other than display or inquiry commands. *Redirect* is a pipeline or a redirection to a file (e.g., `> file` or `| grep ...`). The IBM response is written to the pipeline or file. If *redirect* is not present, the response is written to the standard output of *rjestat*.

An interrupt signal (DEL or BREAK) will cancel the command in progress and cause *rjestat* to return to the command input mode.

**EXAMPLE**

The following command reports the status of all the card readers attached to host A, remote 5. JES2 is assumed.

```
rjestat -cA '$du,rmt5 | grep RD'
```

**DIAGNOSTICS**

The message "RJE error: ..." indicates that *rjestat* found an inconsistency in the RJE system. This may be transient but should be reported to the site administrator.

**FILES**

`/usr/rje/lines` RJE configuration file

`resp`            host response file that exists in the RJE subsystem directory (e.g., `/usr/rje1`).

**SEE ALSO**

`send(1C)`.

*OS/VS2 HASP II Version 4 Operator's Guide*, IBM SRL #GC27-6993.

*Operator's Library: OS/VS2 Reference (JES2)*, IBM SRL #GC38-0210.

**NAME**

*rm*, *rmdir* — remove files or directories

**SYNOPSIS**

*rm* [ *-fri* ] file ...

*rmdir* dir ...

**DESCRIPTION**

*Rm* removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with *y* the file is deleted, otherwise the file remains. No questions are asked when the *-f* option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument *-r* has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the *-i* (interactive) option is in effect, *rm* asks whether to delete each file, and, under *-r*, whether to examine each directory.

*Rmdir* removes entries for the named directories, which must be empty.

**SEE ALSO**

*unlink(2)*.

**DIAGNOSTICS**

Generally self-explanatory. It is forbidden to remove the file *..* merely to avoid the antisocial consequences of inadvertently doing something like:

```
rm -r .*
```

**NAME**

`rmdel` — remove a delta from an SCCS file

**SYNOPSIS**

`rmdel` *-rSID* files

**DESCRIPTION**

*Rmdel* removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see *get(1)*) exists for the named SCCS file, the specified must *not* appear in any entry of the *p-file*).

If a directory is named, *rmdel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s.*) and unreadable files are silently ignored. If a name of *-* is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

**FILES**

x-file        (see *delta(1)*)  
z-file        (see *delta(1)*)

**SEE ALSO**

*delta(1)*, *get(1)*, *help(1)*, *prs(1)*, *scsfile(4)*.

*Source Code Control System User's Guide* in the *UNIX System User's Guide*.

**DIAGNOSTICS**

Use *help(1)* for explanations.

**NAME**

**sact** — print current SCCS file editing activity

**SYNOPSIS**

**sact** files

**DESCRIPTION**

*Sact* informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the *-e* option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of *-* is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

- |         |                                                                                                                          |
|---------|--------------------------------------------------------------------------------------------------------------------------|
| Field 1 | specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta. |
| Field 2 | specifies the SID for the new delta to be created.                                                                       |
| Field 3 | contains the logname of the user who will make the delta (i.e. executed a <i>get</i> for editing).                       |
| Field 4 | contains the date that <i>get -e</i> was executed.                                                                       |
| Field 5 | contains the time that <i>get -e</i> was executed.                                                                       |

**SEE ALSO**

*delta*(1), *get*(1), *unget*(1).

**DIAGNOSTICS**

Use *help*(1) for explanations.

**NAME**

sadp — disk access profiler

**SYNOPSIS**

sadp [ **-th** ] [ **-d** device[**-drive**] ] § [ **n** ]

**DESCRIPTION**

*Sadp* reports disk access location and seek distance, in tabular or histogram form. It samples disk activity once every second during an interval of *s* seconds. This is done repeatedly if *n* is specified. Cylinder usage and disk distance are recorded in units of eight cylinders.

Valid values of *device* are **rp06**, **rm05**, and **disk**. *Drive* specifies the disk drives and it may be:

a drive number in the range supported by *device*,  
two numbers separated by a minus (indicating an inclusive range),

or

a list of drive numbers separated by commas.

Up to eight disk drives may be reported. The **-d** option may be omitted, if only one *device* is present.

The **-t** flag causes the data to be reported in tabular form. The **-h** flag produces a histogram on the printer of the data. Default is **-t**.

**EXAMPLE**

The command:

```
sadp -d rp06 -0 900 4
```

will generate 4 tabular reports, each describing cylinder usage and seek distance of rp06 disk drive 0 during a 15 minute interval.

**FILES**

/dev/kmem



## NAME

sag — system activity graph

## SYNOPSIS

sag [ options ]

## DESCRIPTION

*Sag* graphically displays the system activity data stored in a binary data file by a previous *sar*(1) run. Any of the *sar* data items may be plotted singly, or in combination; as cross plots, or versus time. Simple arithmetic combinations of data may be specified. *Sag* invokes *sar* and finds the desired data by string-matching the data column header (run *sar* to see what's available). These *options* are passed thru to *sar*:

- s *time* Select data later than *time* in the form hh[:mm]. Default is 08:00.
- e *time* Select data up to *time*. Default is 18:00.
- i *sec* Select data at intervals as close as possible to *sec* seconds.
- f *file* Use *file* as the data source for *sar*. Default is the current daily data file `/usr/adm/sa/sadd`.

Other options:

- T *term* Produce output suitable for terminal *term*. See *tplot*(1G) for known terminals. If *term* is *vpr*, output is processed by *vpr* —p and queued to a Versatec printer. Default for *term* is \$TERM.
- x *spec* x axis specification with *spec* in the form:  
"name[op name]...[lo hi]"
- y *spec* y axis specification with *spec* in the same form as above.

*Name* is either a string that will match a column header in the *sar* report, with an optional device name in square brackets, e.g., `r+w/s[dsk-1]`, or an integer value. *Op* is + - \* or / surrounded by blanks. Up to five names may be specified. Parentheses are not recognized. Contrary to custom, + and - have precedence over \* and /. Evaluation is left to right. Thus `A / A + B * 100` is evaluated  $(A/(A+B))*100$ , and `A + B / C + D` is  $(A+B)/(C+D)$ . *Lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, *time* is used. Up to 5 *spec*'s separated by ; may be given for —y. Enclose the —x and —y arguments in "" if blanks or \<CR> are included. The —y default is:

—y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

## EXAMPLES

To see today's CPU utilization:

```
sag
```

To see activity over 15 minutes of all disk drives:

```
TS=`date +%H:%M`
sar -o tempfile 60 15
TE=`date +%H:%M`
sag -f tempfile -s $TS -e $TE -y "r+w/s[dsk]"
```

## FILES

`/usr/adm/sa/sadd` daily data file for day *dd*.

## SEE ALSO

*sar*(1), *tplot*(1G).

**NAME**

sar — system activity reporter

**SYNOPSIS**

sar [-ubdycwaqvmA] [-o file] t [ n ]

sar [-ubdycwaqvmA] [-s time] [-e time] [-i sec] [-f file]

**DESCRIPTION**

*Sar*, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the *-o* option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, *sar* extracts data from a previously recorded *file*, either the one specified by *-f* option or, by default, the standard system activity daily data file */usr/adm/sa/sadd* for the current day *dd*. The starting and ending times of the report can be bounded via the *-s* and *-e time* arguments of the form *hh[:mm[:ss]]*. The *-i* option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

- u Report CPU utilization (the default):  
%usr, %sys, %wio, %idle — portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.
- b Report buffer activity:  
bread/s, bwrit/s — transfers per second of data between system buffers and disk or other block devices;  
lread/s, lwrit/s — accesses of system buffers;  
%cache, %wcache — cache hit ratios, e. g., 1 — bread/lread;  
pread/s, pwrit/s — transfers via raw (physical) device mechanism.
- d Report activity for each block device, e. g., disk or tape drive:  
%busy, avque — portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;  
r+w/s, blks/s — number of data transfers from or to device, number of bytes transferred in 512 byte units;  
await, avserv — average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).
- y Report TTY device activity:  
rawch/s, canch/s, outch/s — input character rate, input character rate processed by canon, output character rate;  
rcvin/s, xmtin/s, mdmin/s — receive, transmit and modem interrupt rates.
- c Report system calls:  
scall/s — system calls of all types;  
sread/s, swrit/s, fork/s, exec/s — specific system calls;  
rchar/s, wchar/s — characters transferred by read and write system calls.
- w Report system swapping and switching activity:  
swpin/s, swpot/s, bswin/s, bswot/s — number of transfers and number of 512 byte units transferred for swapins (including initial loading of some programs) and swapouts;  
pswch/s — process switches.
- a Report use of file access system routines:  
iget/s, namei/s, dirblk/s.
- q Report average queue length while occupied, and % of time occupied:  
runq-sz, %runocc — run queue of processes in memory and runnable;

- swpq-sz, %swpocc — swap queue of processes swapped out but ready to run.
- v Report status of text, process, inode and file tables:  
text-sz, proc-sz, inod-sz, file-sz — entries/size for each table, evaluated once at sampling point;  
text-ov, proc-ov, inod-ov, file-ov — overflows occurring between sampling points.
  - m Report message and semaphore activities:  
msg/s, sema/s — primitives per second.
  - A Report all data. Equivalent to `-udqbwcaymm`.

**EXAMPLES**

To see today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
sar -o temp 60 10
```

To later review disk and tape activity from that period:

```
sar -d -f temp
```

**FILES**

`/usr/adm/sa/sadd` daily data file, where *dd* are digits representing the day of the month.

**SEE ALSO**

`sag(1G)`.

`sar(1M)` in the *UNIX System Administrator's Manual*.

**NAME**

`scat` - concatenate and print files on synchronous printer

**SYNOPSIS**

`scat [ -u ] [ -s ] file ...`

**DESCRIPTION**

*Scat* reads each *file* in sequence and writes it on the standard output, which is assumed to be a synchronous printer device. Thus:

```
scat file > /dev/sp0
```

prints the file, and:

```
cat file1 file2 > /dev/sp0
```

concatenates *file1* and *file2* and places the result on the printer.

If no input file is given, or if the argument `-` is encountered, *scat* reads from the standard input file. Output is buffered in 512-byte blocks unless the `-u` option is specified. The `-s` option makes *scat* silent about non-existent files.

**SEE ALSO**

`cp(1)`, `pr(1)`, `stty(1)`.

**WARNINGS**

*Scat* uses synchronous printers in line mode with the wrap around option enabled. This means that the maximum line length is 79 characters; longer lines will be wrapped back to the beginning of the next line each time the end of a printer line is reached.

**NAME**

scc — C compiler for stand-alone programs

**SYNOPSIS**

scc [ +[ lib ] ] [ option ] ... [ file ] ...

**DESCRIPTION**

*Scc* prepares the named files for stand-alone execution. The *option* and *file* arguments may be anything that can legally be used with the *cc* command; it should be noted, though, that the *-p* (profiling) option, as well as any object module that contains system calls, will cause the executable not to run.

*Scc* defines the compiler constant, *STANDALONE*, so that sections of C programs may be compiled conditionally when the executable will be run stand-alone.

The first argument specifies an auxiliary library that defines the device configuration of the PDP-11 computer for which the stand-alone executable is being prepared. *Lib* may be one of the following:

- A      RP04/05/06 disk and TU16 magnetic tape, or equivalent on the PDP-11 plus RM05 and RM80 disks, and TU78 and TS11 tapes, or equivalent on the VAX
- B      RK11/RK05 disk, RP11/RP03 disk, and TM11/TU16 magnetic tape, or equivalent

If no *+lib* argument is specified, *+A* is assumed. If the *+* argument is specified alone, no configuration library is loaded unless the user supplies his own.

**FILES**

|                  |                          |               |
|------------------|--------------------------|---------------|
| /lib/crt2.o      | execution start-off      |               |
| /usr/lib/lib2.a  | stand-alone library      |               |
| /usr/lib/lib2A.a | +A configuration library | (PDP-11 only) |
| /usr/lib/lib2B.a | +B configuration library | (PDP-11 only) |

**SEE ALSO**

cc(1), ld(1), a.out(4).

**NAME**

`sccsdiff` — compare two versions of an SCCS file.

**SYNOPSIS**

`sccsdiff -rSID1 -rSID2 [-p] [-sn] files`

**DESCRIPTION**

*Sccsdiff* compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

- `-rSID?` *SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.
- `-p` pipe output for each file through *pr*(1).
- `-sn` *n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

**FILES**

`/tmp/get?????` Temporary files

**SEE ALSO**

*bdiff*(1), *get*(1), *help*(1), *pr*(1).

*Source Code Control System User's Guide UNIX System User's Guide.*

**DIAGNOSTICS**

“*file*: No differences” If the two versions are the same.  
Use *help*(1) for explanations.

**NAME**

sdb — symbolic debugger

**SYNOPSIS**

sdb [-w] [-W] [ objfil [ corfil [ directory ] ] ]

**DESCRIPTION**

*Sdb* is a symbolic debugger which can be used with C and F77 programs. It may be used to examine their files and to provide a controlled environment for their execution.

*Objfil* is normally an executable program file which has been compiled with the *-g* (debug) option; if it has not been compiled with the *-g* option, or if it is not an executable file, the symbolic capabilities of *sdb* will be limited, but the file can still be examined, and the program debugged. The default for *objfil* is *a.out*. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is *core*. The core file need not be present. A *-* in place of *corfil* will force *sdb* to ignore any core image file. Source file names in *objfil* are interpreted relative to *directory*.

It is useful to know that at any time there is a *current line* and *current file*. If *corfil* exists then they are initially set to the line and file containing the source statement at which the process terminated or stopped. Otherwise, they are set to the first line in *main()*. The current line and file may be changed with the source file examination commands.

Normally, warnings are provided if the source files used in producing *objfil* cannot be found, or are newer than *objfil*. This checking feature and the accompanying warnings may be disabled by the use of the *-W* flag.

Names of variables are written just as they are in C or F77. Variables local to a procedure may be accessed using the form *procedure:variable*. If no procedure name is given, the procedure containing the current line is used by default. F77 common variables are regarded as local symbols by *sdb*, as the symbolic names are local to procedures.

It is also possible to refer to structure members as *variable.member*, pointers to structure members as *variable->member* and array elements as *variable[number]*. Pointers may be dereferenced by using the form *pointer[0]*. Combinations of these forms may also be used. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by *sdb*. An unqualified structure variable may also be used with various commands. Generally, *sdb* will interpret a structure as a set of variables. Thus, *sdb* will display the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value *sdb* displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as *variable[number][number]...*, or as *variable[number,number,...]*. In place of *number*, the form *number;number* may be used to indicate a range of values, *\** may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, *sdb* displays all the values of an array or section of an array if trailing subscripts are omitted. It displays only the address of the array itself or section specified by the user if subscripts are omitted. A multidimensional parameter in an F77 program cannot be displayed as an array, but it is actually a pointer, whose value is the location of the array. The array itself can be accessed symbolically from the

calling function.

A particular instance of a variable on the stack may be referenced by using the form *procedure:variable,number*. All the variations mentioned in naming variables may be used. *Number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal or hexadecimal.

Line numbers in the source program are referred to as *file-name:number* or *procedure:number*. In either case the number is relative to the beginning of the file. If no procedure or file name is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under *sdb* all addresses refer to the executing program; otherwise they refer to *objfil* or *corfil*. An initial argument of *-w* permits overwriting locations in *objfil*.

### Addresses.

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*) and the *file address* corresponding to a written *address* is calculated as follows:

$$b1 \text{ address} < e1$$

$$\text{file address} = \text{address} + f1 - b1$$

otherwise

$$b2 \text{ address} < e2$$

$$\text{file address} = \text{address} + f2 - b2,$$

otherwise, the requested *address* is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap.

The initial setting of both mappings is suitable for normal *a.out* and *core* files. If either file is not of the kind expected then, for that file, *b1* is set to 0, *e1* is set to the maximum file size and *f1* is set to 0; in this way the whole file can be examined with no address translation.

In order for *sdb* to be used on large files all appropriate values are kept as signed 32 bit integers.

### Commands.

The commands for examining data in the program are:

- t Print a stack trace of the terminated or stopped program.
- T Print the top line of the stack trace.

*variable /lm*

Print the value of *variable* according to length *l* and format *m*. If *l* and *m* are omitted, *sdb* chooses a length and format suitable for the variable's type as declared in the program. The length specifiers are:

- b one byte
- h two bytes (half word)
- l four bytes (long word)

*number*

string length for formats *s* and *a*



Legal values for *m* are:

|          |                                                                                                                     |
|----------|---------------------------------------------------------------------------------------------------------------------|
| <b>c</b> | character                                                                                                           |
| <b>d</b> | decimal                                                                                                             |
| <b>u</b> | decimal, unsigned                                                                                                   |
| <b>o</b> | octal                                                                                                               |
| <b>x</b> | hexadecimal                                                                                                         |
| <b>f</b> | 32 bit single precision floating point                                                                              |
| <b>g</b> | 64 bit double precision floating point                                                                              |
| <b>s</b> | Assume <i>variable</i> is a string pointer and print characters starting at the address pointed to by the variable. |
| <b>a</b> | Print characters starting at the variable's address. This format may not be used with register variables.           |
| <b>p</b> | pointer to procedure                                                                                                |
| <b>i</b> | disassemble machine language instruction with addresses printed symbolically.                                       |
| <b>I</b> | disassemble machine language instruction with addresses just printed numerically.                                   |

The length specifiers are only effective with the formats *d*, *u*, *o* and *x*. If one of these formats is specified and *l* is omitted, the length defaults to the word length of the host machine; 4 for the 3B20S and VAX-11/780. If a numeric length specifier is used for the *s* or *a* command then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command *./*.

The *sh*(1) metacharacters *\** and *?* may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, both variables local to the current procedure and global variables are matched, while if a procedure name is specified then only variables local to that procedure are matched. To match only global variables, the form *:pattern* is used.

*linenumber?lm*  
*variable?lm*

Print the value at the address from *a.out* or *I* space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is 'i'.

*variable=lm*  
*linenumber=lm*  
*number=lm*

Print the address of *variable* or *linenumber*, or the value of *number* in the format specified by *lm*. If no format is given, then *lx* is used. The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

*variable!value*

Set *variable* to the given *value*. The value may be a number, character constant or a variable. The value must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted 'character'. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The *variable* may be an expression which indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type int. C conventions are used in performing any type conversions necessary to perform the indicated assignment.

- x Print the machine registers and the current machine language instruction.
- X Print the current machine language instruction.

The commands for examining source files are:

- e *procedure*
- e *file-name*
- e *directory/*
- e *directory file-name*

The first two forms set the current file to the file containing *procedure* or to *file-name*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, file name, or directory is given, the current procedure and file names are reported.

*/regular expression/*

Search forward from the current line for a line containing a string matching *regular expression* as in *ed(1)*. The trailing */* may be elided.

*?regular expression?*

Search backward from the current line for a line containing a string matching *regular expression* as in *ed(1)*. The trailing *?* may be elided.

- p Print the current line.
- z Print the current line followed by the next 9 lines. Set the current line to the last line printed.
- w Window. Print the 10 lines around the current line.

*number*

Set the current line to the given line number. Print the new current line.

*count +*

Advance the current line by *count* lines. Print the new current line.

*count -*

Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

*count r args*

*count R*

Run the program with the given arguments. The *r* command with no arguments reuses the previous arguments to the program while the *R* command runs the program with no arguments. An argument beginning with *<* or *>* causes redirection for the standard input or output respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

*linenumber c count*

*linenumber C count*

Continue after a breakpoint or interrupt. If *count* is given, it specifies the number of breakpoints to be ignored. *C* continues with the signal which caused the program to stop and *c* ignores it. If a *linenumber* is specified then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes.

*linenumber g count*

Continue after a breakpoint with execution resumed at the given line.

If *count* is given, it specifies the number of breakpoints to be ignored.

*s count*

**S** *count*

Single step. Run the program through *count* lines. If no *count* is given then the program is run for one line. **S** is equivalent to **s** except it steps through subroutine calls.

**i**

**I** Single step by one machine language instruction. **I** steps with the signal which caused the program to stop and **i** ignores it.

*variable* **\$m count**

*address*: **m count**

Single step (as with **s**) until the specified location is modified with a new value. If *count* is omitted, it is effectively infinity. *Variable* must be accessible from the current procedure. Since this command is done by software, it can be very slow.

*level* **v**

Toggle verbose mode, for use when single stepping with **S**, **s** or **m**. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each **C** source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A **v** turns verbose mode off if it is on for any level.

**k** Kill the debugged program.

*procedure*(*arg1, arg2, ...*)

*procedure*(*arg1, arg2, ...*)/*m*

Execute the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to **d**.

*linenumber* **b commands**

Set a breakpoint at the given line. If a procedure name without a line number is given (e.g. "proc:"), a breakpoint is placed at the first line in the procedure even if it was not compiled with the debug flag. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given then execution stops just before the breakpoint and control is returned to *sdb*. Otherwise the *commands* are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If **k** is used as a command to execute at a breakpoint, control returns to *sdb*, instead of continuing execution.

**B** Print a list of the currently active breakpoints.

*linenumber* **d**

Delete a breakpoint at the given line. If no *linenumber* is given then the breakpoints are deleted interactively: Each breakpoint location is printed and a line is read from the standard input. If the line begins with a **y** or **d** then the breakpoint is deleted.

**D** Delete all breakpoints.

**l** Print the last executed line.

*linenumber* **a**

Announce. If *linenumber* is of the form *proc:number*, the command

effectively does a *linenumber* **b l**. If *linenumber* is of the form *proc:*, the command effectively does a *proc: b T*.

Miscellaneous commands:

**!command**

The command is interpreted by *sh*(1).

**new-line**

If the previous command printed a source line then advance the current line by 1 line and print the new current line. If the previous command displayed a core location then display the next core location.

**control-D**

Scroll. Print the next 10 lines of instructions, source or data depending on which was printed last.

**< filename**

Read commands from *filename* until the end of file is reached, and then continue to accept commands from standard input. When *sdb* is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; < may not appear as a command in a file.

**M** Print the address maps.

**M [?/][\*] b e f**

New values for the address map are recorded. The arguments ? and / specify the text and data maps respectively. The first segment, (*b1,e1,f1*) is changed unless \* is specified, in which case the second segment (*b2,e2,f2*) of the mapping is changed. If fewer than three values are given, the remaining map parameters are left unchanged.

**" string**

Print the given string. The C escape sequences of the form *\character* are recognized, where *character* is a nonnumeric character.

**q** Exit the debugger.

The following commands also exist and are intended only for debugging the debugger:

**V** Print the version number.

**Q** Print a list of procedures and files being debugged.

**Y** Toggle debug output.

## FILES

a.out

core

## SEE ALSO

a.out(4), core(4).

## WARNINGS

On the VAX-11/780, C variables are identified internally with an underscore prepended. User variables which differ by only an initial underscore cannot be distinguished, as *sdb* recognizes both internal and external names.

Data which are stored in text sections are indistinguishable from functions.

## BUGS

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

The default type for printing F77 parameters is incorrect. Their address is printed instead of their value.

Tracebacks containing F77 subprograms with multiple entry points may print too many arguments in the wrong order, but their values are correct.

The range of an F77 array subscript is assumed to be 1 to  $n$ , where  $n$  is the dimension corresponding to that subscript. This is only significant when the user omits a subscript, or uses \*, to indicate the full range. There is no problem in general with arrays having subscripts whose lower bounds are not 1.

On the 3B20S there is no hardware trace mode and single stepping is implemented by setting pseudo breakpoints where possible.

The entry point to an optimized function cannot be found on the 3B20S. Setting a breakpoint at the beginning of an optimized function may cause the middle of some instruction within the function to be overwritten. This problem can be circumvented by disassembling the first few instructions of the function, and manually setting a breakpoint at the first instruction after the stack pointer is adjusted.

## NAME

`sdiff` — side-by-side difference program

## SYNOPSIS

`sdiff` [ options ... ] *file1* *file2*

## DESCRIPTION

*Sdiff* uses the output of *diff*(1) 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 *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```

x      |      y
a      |      a
b      <
c      <
d      |      d
      >      c

```

The following options exist:

- `-w n` Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.
- `-l` Only print the left side of any lines that are identical.
- `-s` Do not print identical lines.
- `-o output` Use the next argument, *output*, as the name of a third file that is created as a user controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed 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 editor with the left column
e r    call the editor with the right column
e b    call the editor with the concatenation of left
      and right
e      call the editor with a zero length file
q      exit from the program

```

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

## SEE ALSO

`diff`(1), `ed`(1).

## NAME

*se* — screen editor for video terminals

## SYNOPSIS

*se* [-T[term]] [-ifile] [-ofile] [-s] [file]

## DESCRIPTION

*Se* is an interactive screen editor for use on asynchronous, ASCII CRT terminals. If the *file* argument is given, *se* will read the file into its buffer so that it can be edited. If no *file* is specified, the buffer will be empty and there will be no current file name.

Options to *se* are:

- T Causes *se* to print a list of the terminal types it understands and exit immediately, ignoring all other options.
- Tterm Specifies the terminal type being used. If no -T option is specified, *se* will check the environment variables SETERM and TERM (in that order) to determine the terminal type specified (the first non-null value it finds is the one used). If no terminal type is specified or if the terminal type specified is unknown to *se*, *se* will print a diagnostic followed by a list of terminal types it understands and then exit.
- ifile Causes a sequence of *se* commands to be read from the named *file*. The file is read to end of file. If more than one -i option is given, the files are read in the order specified on the command line. When all -i options have been processed, commands are read from the standard input. A maximum of five files may be specified.
- ofile Causes a copy of all commands given to this invocation of *se* to be placed in *file*. This file may then be used with the -i option.
- s Reduce the number of messages printed on the status line. This is intended for the expert user.

Other than the order of multiple -i options, the order of the options and the filename on the command line is not important.

During editing, *se* displays the contents of the file on the screen. As the file is edited, the screen is updated to reflect changes made in the file contents. If the entire contents of the file will not fit on the screen, *se* displays a portion of it. The limits of the file are indicated on the screen by the TOP OF FILE and BOTTOM OF FILE messages.

The top line of the display is used for a status line. The status line contains (from left to right): the last command entered (or being entered), error messages and the name of the file being edited.

The current position in the file is indicated by the position of the *cursor* on the screen. The cursor can be moved to different file positions by cursor movement commands or find commands. The cursor is not restricted to text already present. If text is inserted or overwritten to the right of the end of the line, the line will be padded with blanks.

*Se* operates in command mode: each character typed is interpreted as part of an *se* command. As each command is recognized, the appropriate action is performed. To add new text to the file, the *insert* command is used. During insert, characters typed are interpreted as text to be added to the file. The text is added before the current cursor position. For example, if the cursor is positioned on the first *r* in the word *edr-formatter* and the

insert command is given, typing *ito* and ending the insert yields **editor-formatter**.

### COMMAND SYNTAX

Most *se* commands are of the form:

[count] [text-identifier] command

The *count* is an optional field, an integer between 1 and 32,767. The default value for *count* is one. The optional *text-identifier* specifies the block of text of interest. Valid text-identifiers are described below; the default value for text-identifiers is dependent on the command. If more than one *count* or *text-identifier* is used, all but the last will be ignored. *Commands* are specified below.

### TEXT IDENTIFIERS

The valid text-identifiers (*text-id*) are:

| Text-id  | Text Represented                         |
|----------|------------------------------------------|
| .        | Character                                |
| w        | Word                                     |
| F        | File                                     |
| l        | Line                                     |
| S (or s) | Screen                                   |
| e        | Previously defined region                |
| /        | Region found by last <b>find</b> command |

In general, a *text-id* block is identified as that in which the cursor is positioned. A *text-id* may also be identified by a cursor positioned on the white space following the *text-id*.

### CURSOR KEYS

The cursor keys on the terminal keyboard are used to move the cursor around the screen and through the file. For terminals with no cursor keys, the **ctrl+z**, **ctrl+x**, **ctrl+c**, **ctrl+v** keys may be used instead of ←, ↓, ↑ and → respectively.

### NOTATION

In the list of *se* commands below, the following notations apply:

|                 |                                                                       |
|-----------------|-----------------------------------------------------------------------|
| [ ]             | items within brackets are optional                                    |
| { }             | one of the items within the braces must be used                       |
| text-id         | identifies a block of text                                            |
| chars           | any string of characters                                              |
| position-cursor | a sequence of <i>cursor-moves</i> or <i>find</i> commands (see below) |

### TEXT COMMANDS

Commands longer than one character (for example, **READ**) may be invoked by typing an unique initial substring followed by a RETURN (*newline*). If the substring is not unique the RETURN is ignored. The **BREAK** key causes *se* to stop its current action and return to *its* command level.

#### cursor moves

[count] cursor key Move the cursor *count* lines up (↑) or down (↓) or *count* characters to the left (←) or the right (→). Screen scroll will occur if the top or bottom of screen is encountered. The cursor will wrap at line beginning and end as expected.

[count] [text-id] cursor key Move the cursor the specified amount of *text-id* blocks. If the *text-id* is character (.) (default),



the action is the same as for plain cursor key use (see above). For all other *text-ids*, ← means *beginning of*, → means *end of*, ↑ means *previous*, and ↓ means *next*. For example, S↓ means go to the next screen.

|                  |                                                                                                                  |
|------------------|------------------------------------------------------------------------------------------------------------------|
| <b>space-bar</b> | The space-bar moves the cursor one character to the right (equivalent to .→).                                    |
| <b>RETURN</b>    | The RETURN key moves the cursor to the beginning of the next line.                                               |
| <b>TAB</b>       | The TAB key moves the cursor to the next tab position (set every 8 columns).                                     |
| <b>HOME</b>      | For terminals that have a HOME key, it moves the cursor to the top left corner of the screen (equivalent to S←). |

### Define Region

**b** [position-cursor] **ctrl+d** Define an arbitrary linear region. Any command that changes the file being edited will cause the current region to be undefined.

### Copy text

[count] [text-id] **c** [position-cursor] **ctrl+d** Copy *text-id* block (default is one character) at new cursor position.

### Delete text

[count] [text-id] **d** Delete *text-id* block (default is one character).

### Refresh document display

**DISPLAY** Rewrites display from the file. Useful to restore contents of screen from the effects of line noise etc.

### Edit file

**EDIT** [filename] { **ctrl+d**, **RETURN** } Start editing the specified file. If no file name has been specified, use the current file. If the contents of the current file have been altered since the last **WRITE** command, the user is first queried as to whether to save those changes.

### Find string occurrence

[text-id] **f** chars { **ctrl+d**, **RETURN** } Search *text-id* (default is entire file) for *chars* and position cursor there. The cursor is not moved if *chars* are not found. The chars are interpreted as a regular expression (see *regexp(5)*).

### Find all and execute command automatically

[count] [text-id] **g** chars { **ctrl+d**, **RETURN** } *command* Search *text-id* (default is entire file) for all occurrences of *chars*; position-cursor at first occurrence and execute *command*. Continue to next occurrence and apply the same *command*, and so on. The *command* may not be another global command. The chars are interpreted as a regular expression (see *regexp(5)*).

**Find all and execute command interactively**

[count] [text-id] G chars { ctrl+d, RETURN } command

Search *text-id* (default is entire file) for first occurrence of chars; position-cursor at first occurrence and wait for *command*; execute *command* and continue to next occurrence where a new *command* may be input, and so on. The *command* may not be another global command. The chars are interpreted as a regular expression (see *regexp(5)*).

**Insert text**

[text-id] i chars ctrl+d

Insert text at the current cursor position. If the *text-id* is 1, a blank line is inserted and the cursor positioned at the beginning of that line. Use of cursor-keys (no preceding *count* or *text-id*) positions the cursor at the next character to be inserted. The back-space key will cause the previous character to be deleted.

**Move text**

[count] [text-id] m [position-cursor] ctrl+d

Reposition *text-id* block (default is one character) at new position. It is an error if the new position is within the text to be moved.

**Overwrite text**

o chars ctrl+d

Performs one-to-one character replacement beginning at cursor position. Use of cursor-keys (no preceding *count* or *text-id*) positions the cursor at the next character to be overwritten. The back-space key will cause the previous character to be deleted.

**Leave the editor**

q

Exits from *se*. If the contents of the current file have been altered since the last WRITE command, the user is first queried as whether to save those changes.

**Get text**

READ [filename] { ctrl+d, RETURN }

Insert text from *filename* at cursor position. If no *filename* is specified, the current filename is used. The cursor position is unchanged.

**Replace text**

[count] [text-id] r chars ctrl+d

Replace *text-id* block (default is one character) with text.

**Undo last command**

UNDO

Undoes last text-modifying command. An UNDO may not be undone.

**Save text**

[count] [text-id] WRITE [filename] { ctrl+d, RETURN }

Save text from *text-id* (default is entire file) in the named file. If *filename* is not specified, text is saved in the file currently being edited. Note that existing text in the file is replaced.

**Process through UNIX**

[count] [text-id] X UNIX-command { ctrl+d, RETURN }

Passes *text-id* block (default is no text) to the *UNIX-command* as standard input and replaces *text-id* block with the standard output from the *UNIX-command*.

**Request help**

?

Display a listing of available *se text-ids*, commands and their syntax.

**Escape from editor**

[count] [text-id] ! UNIX-command { ctrl+d, RETURN }

If the *text-id* or *count* is specified, it is given as standard input to the UNIX command. Otherwise, standard input is the same as for *se*. No changes are made to the file being edited.

**Repeat last command**

.

Ditto repeats the last command. This means the command plus preceding *text-id* and *count*.

**Go to line**

N #

Move to line N, where N is an integer between 1 and 32,767.

**Erase input**

@

Cause *se* to ignore any partially typed command (including count, modifier, and multi-character command).

**TERMINAL REQUIREMENTS**

*Se* can run on any terminal with suitable cursor addressing. In order to use cursor keys, they must emit characters to the host computer. Performance may be degraded if the terminal does not have:

- character insert and delete
- line insert and delete
- erase to end of line and page

If the terminal type specified is not suitable (i.e. it has no cursor addressing), *se* prints a diagnostic and exits immediately.

The environment variable **TERMINFO** modifies the search for the specified terminal type in the terminal description file. If present, it should contain one of two kinds of values:

- an alternate file name for the terminal description file (in this case, the first character must be a /). This file will be used to search for a description of the specified terminal instead of the default terminal description file.
- the description for a specific terminal (this should be the entry from the terminal description file with the escaped newlines removed). This description will be treated as though it had been prepended to the default terminal description file. Using **TERMINFO** in this manner allows the redefinition of a specific terminal description or the inclusion of a description for a terminal that is not included in the default terminal description file.

If the description contained in **TERMINFO** is that of the terminal to be used with *se*, start-up time for *se* can be reduced considerably since the terminal description file need not be searched.

**FILES**

/tmp/se#           temporary; # is the process number.  
/tmp/sei#          record of keystrokes; # is the process number.  
/usr/lib/se.term   terminal description file

**DIAGNOSTICS**

Error messages are displayed on the message line on the screen during editing.

**WARNING**

Regular expressions span more than one line, thus *abc.\*xyz* may match the entire file.

Some terminals need persuasion to make the cursor keys emit characters. For example, HP2621 cursor keys only emit characters when the function labels are displayed and the SHIFT key is held down and the cursor key struck.

**SEE ALSO**

regexp(5).

## NAME

sed — stream editor

## SYNOPSIS

sed [ *-n* ] [ *-e* script ] [ *-f* sfile ] [ files ]

## DESCRIPTION

*Sed* copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The *-f* option causes the script to be taken from file *sfile*; these options accumulate. If there is just one *-e* option and no *-f* options, the flag *-e* may be omitted. The *-n* option suppresses the default output. A script consists 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), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under *-n*) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **\$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of *ed(1)* modified thus:

In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address *\xabc\xdfx*, the second *x* stands for itself, so that the regular expression is *abcxdef*.

The escape sequence *\n* matches a new-line *embedded* in the pattern space.

A period *.* matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with **\** to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an *s* command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* 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*. If *label* is empty, branch to the end of the script.
- (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 new-line. 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 new-line. (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 new-line to the standard output.
- (1) **q** Quit. Branch to the end of the script. Do not start a new cycle.
- (2) **r** *rfile* Read the contents of *rfile*. 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:
- 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** *wfile* Write. Append the pattern space to *wfile* 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** *wfile*  
Write. Append the pattern space to *wfile*.
- (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***

Don't. 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.

**(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.

**(0)** An empty command is ignored.

**SEE ALSO**

awk(1), ed(1), grep(1).

**NAME**

*send*, *gath* — gather files and/or submit RJE jobs

**SYNOPSIS**

*gath* [-ih] file ...

*send* argument ...

**DESCRIPTION****Gath**

*Gath* concatenates the named files and writes them to the standard output. Tabs are expanded into spaces according to the format specification for each file (see *fspec*(4)). The size limit and margin parameters of a format specification are also respected. Non-graphic characters other than tabs are identified by a diagnostic message and excised. The output of *gath* contains no tabs unless the *-h* flag is set, in which case the output is written with standard tabs (every eighth column).

Any line of any of the files which begins with *~* is interpreted by *gath* as a control line. A line beginning "*~* " (tilde,space) specifies a sequence of files to be included at that point. A line beginning "*~!*" specifies a UNIX command; that command is executed, and its output replaces the "*~!*" line in the *gath* output.

Setting the *-i* flag prevents control lines from being interpreted and causes them to be output literally.

A file name of *-* at any point refers to standard input, and a control line consisting of *~* is a logical EOF. Keywords may be defined by specifying a replacement string which is to be substituted for each occurrence of the keyword. Input may be collected directly from the terminal, with several alternatives for prompting. In fact, all of the special arguments and flags recognized by the *send* command are also recognized and treated identically by *gath*. Several of them only make sense in the context of submitting an RJE job.

**Send**

*Send* is a command-level interface to the RJE subsystems. It allows the user to collect input from various sources in order to create a run stream consisting of card images, and submit this run stream for transmission to an IBM host computer. Output from the IBM system may be returned to the user in either ASCII text form or EBCDIC punch format (see *pnch*(4)).

Possible sources of input to *send* are: ordinary files, standard input, the terminal, and the output of a command or shell file. Each source of input is treated as a virtual file, and no distinction is made based upon its origin. Typical input is an ASCII text file of the sort that is created by the editor *ed*(1). An optional format specification appearing in the first line of a file (see *fspec*(4)) determines the settings according to which tabs are expanded into spaces. In addition, lines that begin with *~* are normally interpreted as commands controlling the execution of *send*. They may be used to set or reset flags, to define keyword substitutions, and to open new sources of input in the midst of the current source. Other text lines are translated one-for-one into card images of the run stream.

The run stream that results from this collection is treated as one job by the RJE subsystems. *Send* prints the card count of the run stream, and the queuer that is invoked prints the name of the temporary file that holds the job while it is awaiting transmission. The initial card of a job submitted to a host must have a // in the first column. Any cards preceding this card will be excised. If a host computer is not specified before the first card of



the runstream is ready to be sent, *send* will select a reasonable default. All cards beginning with */\*\$* will be excised from the runstream, because they are HASP command cards.

The arguments that *send* accepts are described below. An argument is interpreted according to the first pattern that it matches. Preceding a character with *\* causes it to lose any special meaning it might otherwise have when matching against an argument pattern.

|                                  |                                                                                                                                                                                                     |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .                                | Close the current source.                                                                                                                                                                           |
| -                                | Open standard input as a new source.                                                                                                                                                                |
| +                                | Open the terminal as a new source.                                                                                                                                                                  |
| : <i>spec</i> :                  | Establish a default format specification for included sources,<br>e.g., <i>m6t-12</i> :                                                                                                             |
| : <i>message</i>                 | Print message on the terminal.                                                                                                                                                                      |
| -: <i>prompt</i>                 | Open standard input and, if it is a terminal, print <i>prompt</i> .                                                                                                                                 |
| +: <i>prompt</i>                 | Open the terminal and print <i>prompt</i> .                                                                                                                                                         |
| - <i>flags</i>                   | Set the specified flags, which are described below.                                                                                                                                                 |
| + <i>flags</i>                   | Reset the specified flags.                                                                                                                                                                          |
| = <i>flags</i>                   | Restore the specified flags to their state at the previous level.                                                                                                                                   |
| ! <i>command</i>                 | Execute the specified UNIX <i>command</i> via the one-line shell, with input redirected to <i>/dev/null</i> as a default. Open the standard output of the command as a new source.                  |
| \$ <i>line</i>                   | Collect contiguous arguments of this form and write them as consecutive lines to a temporary file; then have the file executed by the shell. Open the standard output of the shell as a new source. |
| @ <i>directory</i>               | The current directory for the send process is changed to <i>directory</i> . The original directory will be restored at the end of the current source.                                               |
| ~ <i>comment</i>                 | Ignore this argument.                                                                                                                                                                               |
| ?: <i>keyword</i>                | Prompt for a definition of <i>keyword</i> from the terminal unless <i>keyword</i> has an existing definition.                                                                                       |
| ? <i>keyword</i> = <i>^xx</i>    | Define the <i>keyword</i> as a two digit hexadecimal character code unless it already has a non null replacement.                                                                                   |
| ? <i>keyword</i> = <i>string</i> | Define the <i>keyword</i> in terms of a replacement string unless it already has a non null replacement.                                                                                            |
| =: <i>keyword</i>                | Prompt for a definition of <i>keyword</i> from the terminal.                                                                                                                                        |
| <i>keyword</i> = <i>^xx</i>      | Define <i>keyword</i> as a two-digit hexadecimal character code.                                                                                                                                    |
| <i>keyword</i> = <i>string</i>   | Define <i>keyword</i> in terms of a replacement string.                                                                                                                                             |

*host* The host machine that the job should be submitted to. It can be any name that corresponds to one in the first column of the RJE configuration file (*/usr/rje/lines*).

*file-name* Open the specified file as a new source of input.

When commands are executed via *\$* or *!* the shell environment (see *environ(5)*) will contain the values of all send keywords that begin with *\$* and have the syntax of a shell variable.

The flags recognized by *send* are described in terms of the special processing that occurs when they are set:

- l List card images on standard output. EBCDIC characters are translated back to ASCII.
- q Do not output card images.
- f Do not fold lower case to upper.
- t Trace progress on diagnostic output, by announcing the opening of input sources.
- k Ignore the keywords that are active at the previous level and erase any keyword definitions that have been made at the current level.
- r Process included sources in raw mode; pack arbitrary 8-bit bytes one per column (80 columns per card) until an EOF.
- i Do not interpret control lines in included sources; treat them as text.
- s Make keyword substitutions before detecting and interpreting control lines.
- y Suppress error diagnostics and submit job anyway.
- g Gather mode, qualifying *-l* flag; list text lines before converting them to card images.
- h Write listing with standard tabs.
- p Prompt with *\** when taking input from the terminal.
- m When input returns to the terminal from a lower level, repeat the prompt, if any.
- a Make *-k* flag propagate to included sources, thereby protecting them from keyword substitutions.
- c List control lines on diagnostic output.
- d Extend the current set of keyword definitions by adding those active at the end of included sources.
- x This flag guarantees that the job will be transmitted in the order of submission (relative to other jobs sent with this flag).

Control lines are input lines that begin with *~*. In the default mode *+ir*, they are interpreted as commands to *send*. Normally they are detected immediately and read literally. The *-s* flag forces keyword substitutions to be made before control lines are intercepted and interpreted. This can lead to unexpected results if a control line uses a keyword which is defined within an immediately preceding *~\$* sequence. Arguments appearing in control lines are handled exactly like the command arguments to *send*, except that they are processed at a nested level of input.

The two possible formats for a control line are: “`~argument`” and “`~ argument ...`”. In the first case, where the `~` is not followed by a space, the remainder of the line is taken as a single argument to `send`. In the second case, the line is parsed to obtain a sequence of arguments delimited by spaces. In this case the quotes `'` and `"` may be employed to pass embedded spaces.

The interpretation of the argument `.` is chosen so that an input line consisting of `~.` is treated as a logical EOF. The following example illustrates some of the above conventions:

```
send -
~ argument ...
~.
```

This sequence of three lines is equivalent to the command synopsis at the beginning of this description. In fact, the `-` is not even required. By convention, the `send` command reads standard input if no other input source is specified. `Send` may therefore be employed as a filter with side-effects.

The execution of the `send` command is controlled at each instant by a current environment, which includes the format specification for the input source, a default format specification for included sources, the settings of the mode flags, and the active set of keyword definitions. This environment can be altered dynamically. When a control line opens a new source of input, the current environment is pushed onto a stack, to be restored when input resumes from the old source. The initial format specification for the new source is taken from the first line of the file. If none is provided, the established default is used or, in its absence, standard tabs. The initial mode settings and active keywords are copied from the old environment. Changes made while processing the new source will not affect the environment of the old source, with one exception: if `-d` mode is set in the old environment, the old keyword context will be augmented by those definitions that are active at the end of the new source.

When `send` first begins execution, all mode flags are reset, and the values of the shell environment variables become the initial values for keywords of the same name with a `$` prefixed.

The initial reset state for all mode flags is the `+` state. In general, special processing associated with a mode `N` is invoked by flag `-N` and is revoked by flag `+N`. Most mode settings have an immediate effect on the processing of the current source. Exceptions to this are the `-r` and `-i` flags, which apply only to included source, causing it to be processed in an uninterpreted manner.

A keyword is an arbitrary 8-bit ASCII string for which a replacement has been defined. The replacement may be another string or the hexadecimal code for a single 8-bit byte. At any instant, a given set of keyword definitions is active. Input text lines are scanned, in one pass from left to right, and longest matches are attempted between substrings of the line and the active set of keywords. Characters that do not match are output, subject to folding and the standard translation. Keywords are replaced by the specified hexadecimal code or replacement string, which is then output character by character. The expansion of tabs and length checking, according to the format specification of an input source, are delayed until substitutions have been made in a line.

All of the keywords definitions made in the current source may be deleted by setting the `-k` flag. It then becomes possible to reuse them. Setting the `-k` flag also causes keyword definitions active at the previous source level to be ignored. Setting the `+k` flag causes keywords at the previous level to be ignored but does not delete the definitions made at the current level. The `=k` argument reactivates the definitions of the previous level.

When keywords are redefined, the previous definition at the same level of source input is lost, however the definition at the previous level is only hidden, to be reactivated upon return to that level unless a `-d` flag causes the current definition to be retained.

Conditional prompts for keywords, `?:A,/p` which have already been defined at some higher level to be null or have a replacement will simply cause the definitions to be copied down to the current level; new definitions will not be solicited.

Keyword substitution is an elementary macro facility that is easily explained and that appears useful enough to warrant its inclusion in the *send* command. More complex replacements are the function of a general macro processor (*m4*(1), perhaps). To reduce the overhead of string comparison, it is recommended that keywords be chosen so that their initial characters are unusual. For example, let them all be upper case.

*Send* performs two types of error checking on input text lines. Firstly, only ASCII graphics and tabs are permitted in input text. Secondly, the length of a text line, after substitutions have been made, may not exceed 80 bytes. The length of each line may be additionally constrained by a size parameter in the format specification for an input source. Diagnostic output provides the location of each erroneous line, by line number and input source, a description of the error, and the card image that results. Other routine errors that are announced are the inability to open or write files, and abnormal exits from the shell. Normally, the occurrence of any error causes *send*, before invoking the queuer, to prompt for positive affirmation that the suspect run stream should be submitted.

Before submitting a job to a host, *send* translates 8-bit ASCII characters into their EBCDIC equivalents. The conversion for 8-bit ASCII characters in the octal range 040-176 is based on the character set described in "Appendix H" of *IBM System/370 Principles of Operation* (IBM SRL GA22-7000). Each 8-bit ASCII character in the range 040-377 possesses an EBCDIC equivalent into which it is mapped, with five exceptions: `~` into `~`, 0345 into `~`, 0325 into `¢`, 0313 into `|`, 0177 (DEL) is illegal. In listings requested from *send* and in printed output returned by the subsystem, the reverse translation is made with the qualification that EBCDIC characters that do not have valid 8-bit ASCII equivalents are translated into `~`.

Additional control over the translation process is afforded by the `-f` flag and hexadecimal character codes. As a default, *send* folds lower-case letters into upper case. Setting the `-f` flag inhibits any folding. Non-standard character codes are obtained as a special case of keyword substitution.

#### SEE ALSO

*m4*(1), *rjstat*(1C), *sh*(1), *fspec*(4), *pnch*(4), *ascii*(5), *environ*(5).  
*UNIX Remote Job Entry User's Guide* in the *UNIX System User's Guide*.

**BUGS**

Standard input is read in blocks, and unused bytes are returned via *lseek(2)*. If standard input is a pipe, multiple arguments of the form `-` and `-:prompt` should not be used, nor should the logical EOF (`~`).

## NAME

sh, rsh — shell, the standard/restricted command programming language

## SYNOPSIS

```
sh [ -ceiknrstuvx ] [ args ]
rsh [ -ceiknrstuvx ] [ args ]
```

## DESCRIPTION

*Sh* is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

## Commands.

A *simple-command* is a sequence of non-blank *words* separated by *blanks* (a *blank* is a tab or a space). 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 *exec(2)*). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal(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(2)* to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more 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 (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol && (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

**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 file-name generation (see *File Name Generation* below).

**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 sub-shell.

{*list*;}

*list* is simply executed.

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 **#** causes that word and all the following characters up to a new-line to be ignored.

**Command Substitution.**

The standard output from a command enclosed in a pair of grave accents (**`**) may be used as part or all of a word; trailing new-lines are removed.

**Parameter Substitution.**

The character **\$** is used to introduce substitutable *parameters*. Positional parameters may be assigned values by set. Variables may be set by writing:

*name*=*value* [ *name*=*value* ] ...

Pattern-matching is not performed on *value*.

**\${parameter}**

A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters **\***, **@**, **#**, **?**, **-**, **\$**, and **!**. 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. A *name* must begin with a letter or underscore. If *parameter* is a digit then it is a positional parameter. If *parameter* is **\*** or **@**, then all the positional parameters, starting with **\$1**, are substituted (separated by spaces). Parameter **\$0** is set from argument zero when the shell is invoked.

**\${parameter:-word}**

If *parameter* is set and is non-null then substitute its value; otherwise substitute *word*.

**\${parameter:=word}**

If *parameter* is not set or is null then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

**\${parameter:?word}**

If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, then the message "parameter null or not set" is printed.

**\${parameter:+word}**

If *parameter* is set and is non-null then 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`}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

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). The user may not change PATH if executing under *rsh*.
- CDPATH The search path for the *cd* command.
- MAIL If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file.
- PS1 Primary prompt string, by default "\$ ".
- PS2 Secondary prompt string, by default "> ".
- IFS Internal field separators, normally *space*, *tab*, and *new-line*.

The shell gives default values to PATH, PS1, PS2, and IFS, while HOME and MAIL are not set at all by the shell (although HOME is 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.

#### File Name Generation.

Following substitution, each command *word* is scanned for the characters \*, ?, and [. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. The character . at the start of a file name 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 "!" then 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:

```
; & ( ) | ^ < > new-line space tab
```



A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair \new-line is ignored. All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, ~, ", and \$. "\$\*" is equivalent to "\$1 \$2 ...", whereas "\$@" is equivalent to "\$1" "\$2" ....

### Prompting.

When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of PS2) is issued.

### Input/Output.

Before a command is executed, its 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; substitution occurs before *word* or *digit* is used:

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <word       | Use file <i>word</i> as standard input (file descriptor 0).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| >word       | Use file <i>word</i> as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.                                                                                                                                                                                                                                                                                                                                                                                                    |
| >>word      | Use file <i>word</i> as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.                                                                                                                                                                                                                                                                                                                                                                                          |
| <<[ - ]word | The shell input is read up to a line that is the same as <i>word</i> , or to an end-of-file. The resulting document becomes the standard input. If any character of <i>word</i> is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) \new-line is ignored, and \ must be used to quote the characters \, \$, ~, and the first character of <i>word</i> . If - is appended to <<, then all leading tabs are stripped from <i>word</i> and from the document. |
| <&digit     | The standard input is duplicated from file descriptor <i>digit</i> (see <i>dup</i> (2)). Similarly for the standard output using >.                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <&-         | The standard input is closed. Similarly for the standard output using >.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

If one of the above is preceded by a digit, then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by & then 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.

Redirection of output is not allowed in the restricted shell.

### Environment.

The *environment* (see *environ*(5)) 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. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates

new ones, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, 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 args                and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b** and then **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, with the exception of signal 11 (but see also the **trap** command below).

### Execution.

Each time a command is executed, the above substitutions are carried out. Except for the *Special Commands* listed below, a new process is created and an attempt is made to execute the command via *exec(2)*.

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 **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that 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 the command name contains a **/** then the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a sub-shell.

### Special Commands.

The following commands are executed in the shell process and, except as specified, no input/output redirection is permitted for such commands:

- :** No effect; the command does nothing. A zero exit code is returned.
- . file** Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.
- break [ n ]** Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.
- continue [ n ]** Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n*-th enclosing loop.
- cd [ arg ]** Change the current directory to *arg*. The shell parameter **HOME** is

the default *arg*. The shell parameter CDPATH defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that 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 *arg* begins with a / then the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by *rsh*.

**eval** [ *arg ...* ]

The arguments are read as input to the shell and the resulting command(s) executed.

**exec** [ *arg ...* ]

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, cause the shell input/output to be modified.

**exit** [ *n* ]

Causes a shell to exit with the exit status specified by *n*. If *n* is omitted then the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

**export** [ *name ...* ]

The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, then a list of all names that are exported in this shell is printed.

**newgrp** [ *arg ...* ]

Equivalent to `exec newgrp arg ...`

**read** [ *name ...* ]

One line is read from the standard input and 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*. The return code is 0 unless an end-of-file is encountered.

**readonly** [ *name ...* ]

The given *names* are marked *readonly* and the values of these *names* may not be changed by subsequent assignment. If no arguments are given, then a list of all *readonly* names is printed.

**set** [ `--ekntuvx` [ *arg ...* ] ]

- `-e` Exit immediately if a command exits with a non-zero exit status.
- `-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 flags; useful in setting \$1 to `-`.

Using `+` rather than `-` causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in `$-`. The remaining arguments are positional parameters and are assigned, in order, to `$1`, `$2`, .... If no arguments are given then the values of all names are printed.

**shift** [ *n* ]

The positional parameters from `$n+1 ...` are renamed `$1 ...`. If *n* is not given, it is assumed to be 1.

- test** Evaluate conditional expressions. See *test(1)* for usage and description.
- times** Print the accumulated user and system times for processes run from the shell.
- trap** [ *arg* ] [ *n* ] ...  
*arg* is a command to be read and executed when the shell receives signal(s) *n*. (Note that *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. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent then all trap(s) *n* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by the commands it invokes. If *n* is 0 then 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.
- ulimit** [ *-fp* ] [ *n* ]  
 imposes a size limit of *n*  
 -f imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.  
 -p changes the pipe size to *n* (UNIX/RT only).  
 If no option is given, -f is assumed.
- umask** [ *nnn* ]  
 The user file-creation mask is set to *nnn* (see *umask(2)*). If *nnn* is omitted, the current value of the mask is printed.
- wait** [ *n* ]  
 Wait for the specified process and report its termination status. If *n* is not given then all currently active child processes are waited for and the return code is zero.

### Invocation.

If the shell is invoked through *exec(2)* and the first character of argument zero is -, commands are initially read from */etc/profile* and then 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 */bin/sh*. The flags below are interpreted by the shell on invocation only; Note that unless the -c or -s flag 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 to that command file:

- c *string* If the -c flag is present then commands are read from *string*.
- s If the -s flag is present or if no arguments remain then commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.
- i If the -i flag is present or if the shell input and output are attached to a terminal, then 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.
- r If the -r flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

**Rsh Only.**

*Rsh* is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

- changing directory (see *cd(1)*),
- setting the value of `$PATH`,
- specifying path or command names containing `/`,
- redirecting output (`>` and `>>`).

The restrictions above are enforced after `.profile` is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the `.profile` has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., `/usr/rbin`) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

**EXIT STATUS**

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively then 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).

**FILES**

- `/etc/profile`
- `$HOME/.profile`
- `/tmp/sh*`
- `/dev/null`

**SEE ALSO**

`cd(1)`, `env(1)`, `login(1)`, `newgrp(1)`, `test(1)`, `umask(1)`, `dup(2)`, `exec(2)`, `fork(2)`, `pipe(2)`, `signal(2)`, `ulimit(2)`, `umask(2)`, `wait(2)`, `a.out(4)`, `profile(4)`, `environ(5)`.

**BUGS**

The command `readonly` (without arguments) produces the same output as the command `export`.

If `<<` is used to provide standard input to an asynchronous process invoked by `&`, the shell gets mixed up about naming the input document; a garbage file `/tmp/sh*` is created and the shell complains about not being able to find that file by another name.

**NAME**

*size* — print section sizes of common object files

**SYNOPSIS**

*size* [-o] [-x] [-V] files

**DESCRIPTION**

The *size* command produces section size information for each section in the common object files. The size of the text, data and bss (uninitialized data) sections are printed along with the total size of the object file. If an archive file is input to the *size* command the information for all archive members is displayed.

Numbers will be printed in decimal unless either the *-o* or the *-x* option is used, in which case they will be printed in octal or in hexadecimal, respectively.

The *-V* flag will supply the version information on the *size* command.

**SEE ALSO**

as(1), cc(1), ld(1), a.out(4), ar(4).

**DIAGNOSTICS**

size: name: cannot open  
if *name* cannot be read.

size: name: bad magic  
if *name* is not an appropriate common object file.

**NAME**

size — print sizes of object files

**SYNOPSIS**

size [ object ... ]

**DESCRIPTION**

*Size* prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in octal and decimal, of each object-file argument. If no file is specified, *a.out* is used.

**SEE ALSO**

*a.out*(4).

**NAME**

sleep — suspend execution for an 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**

alarm(2), sleep(3C).

**BUGS**

*Time* must be less than 65536 seconds.



## NAME

sno - SNOBOL interpreter

## SYNOPSIS

sno [ files ]

## DESCRIPTION

*Sno* is a SNOBOL compiler and interpreter (with slight differences). *Sno* obtains input from the concatenation of the named *files* and the standard input. All input through a statement containing the label **end** is considered program and is compiled. The rest is available to **syspit**.

*Sno* differs from SNOBOL in the following ways:

There are no unanchored searches. To get the same effect:

```
a ** b           unanchored search for b.
a *x* b = x c    unanchored assignment
```

There is no back referencing.

```
x = "abc"
a *x* x           is an unanchored search for abc.
```

Function declaration is done at compile time by the use of the (non-unique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and the use of the name **define** is preempted. There is no provision for automatic variables other than parameters. Examples:

```
define f( )
define f(a, b, c)
```

All labels except **define** (even **end**) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on **end** cannot merely name a label.

If **start** is a label in the program, program execution will start there. If not, execution begins with the first executable statement; **define** is not an executable statement.

There are no builtin functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators / and \* must be set off by spaces.

The right side of assignments must be non-empty.

Either ' or \* may be used for literal quotes.

The pseudo-variable **syspnt** is not available.

## SEE ALSO

awk(1).

*SNOBOL, a String Manipulation Language*, by D. J. Farber, R. E. Griswold, and I. P. Polonsky, *JACM* 11 (1964), pp. 21-30.

## NAME

sort — sort and/or merge files

## SYNOPSIS

sort [-cmubdfinrtx] [+pos1 [-pos2]] ... [-o output] [names]

## DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name — means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

- b** Ignore leading blanks (spaces and tabs) in field comparisons.
- d** “Dictionary” order: only letters, digits and blanks are significant in comparisons.
- f** Fold upper case letters onto lower case.
- i** Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.
- n** An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **n** implies option **b**.
- r** Reverse the sense of comparisons.
- tx** “Tab character” separating fields is *x*.

The notation *+pos1 -pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m.n*, optionally followed by one or more of the flags *bdfinr*, where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first non-blank in the field; **b** is attached independently to *pos2*. A missing *.n* means *.0*; a missing *-pos2* means the end of the line. Under the *-tx* option, fields are strings separated by *x*; otherwise fields are non-empty non-blank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

These option arguments are also understood:

- 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** Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.
- o** The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

## EXAMPLES

Print in alphabetical order all the unique spellings in a list of words (capitalized words differ from uncapitalized):

```
sort -u +0f +0 list
```

Print the password file (*passwd*(4)) sorted by user ID (the third colon-separated field):

```
sort -t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month-day) entries (the options `-um` with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort -um +0 -1 dates
```

**FILES**

*/usr/tmp/stm???*

**SEE ALSO**

*comm*(1), *join*(1), *uniq*(1).

**DIAGNOSTICS**

Comments and exits with non-zero status for various trouble conditions and for disorder discovered under option `-c`.

**BUGS**

Very long lines are silently truncated.

**NAME**

spell, hashmake, spellin, hashcheck — find spelling errors

**SYNOPSIS**

```
spell [ -v ] [ -b ] [ -x ] [ -l ] [ +local_file ] [ files ]
/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 spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

*Spell* ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Under the *-v* option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the *-b* option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Under the *-x* option, every plausible stem is printed with = for each word.

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*. Under the *-l* option, *spell* will follow the chains of *all* included files.

Under the *+local\_file* option, 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 spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., *thier=thy+y+ier*) that would otherwise pass.

Three routines 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.

**FILES**

D\_SPELL=/usr/lib/spell/hlist[ab] hashed spelling lists, American & British  
S\_SPELL=/usr/lib/spell/hstop hashed stop list  
H\_SPELL=/usr/lib/spell/spellhist history file  
/usr/lib/spell/spellprog program

**SEE ALSO**

deroff(1), eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(1).

**BUGS**

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling\_list* via *spellin*.

The British spelling feature was done by an American.

## NAME

spline — interpolate smooth curve

## SYNOPSIS

spline [ options ]

## DESCRIPTION

*Spline* takes pairs of numbers from the standard input as abscissas 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., pp. 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph*(1G).

The following *options* are recognized, each as a separate argument:

- 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:  

$$y_0'' = ky_1', \quad y_n'' = ky_{n-1}'$$
 is set by the next argument (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, i.e., 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 abscissas start at lower limit (default 0).

## SEE ALSO

*graph*(1G).

## DIAGNOSTICS

When data is not strictly monotone in  $x$ , *spline* reproduces the input without interpolating extra points.

## BUGS

A limit of 1,000 input points is enforced silently.

**NAME**

split — split a file into pieces

**SYNOPSIS**

split [ *-n* ] [ file [ name ] ]

**DESCRIPTION**

*Split* reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with *aa* appended, and so on lexicographically, up to *zz* (a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, *x* is default.

If no input file is given, or if *-* is given in its stead, then the standard input file is used.

**SEE ALSO**

bfs(1), csplit(1).

**NAME**

*stat* — statistical network useful with graphical commands

**SYNOPSIS**

node-name [options] [files]

**DESCRIPTION**

*Stat* is a collection of command level functions (nodes) that can be interconnected using *sh(1)* to form a statistical network. The nodes reside in */usr/bin/graf* (see *graphics(1G)*). Data is passed through the network as sequences of numbers (vectors), where a number is of the form:

[sign](digits)(.digits)[e[sign]digits]

evaluated in the usual way. Brackets and parentheses surround fields. All fields are optional, but at least one of the fields surrounded by parentheses must be present. Any character input to a node that is not part of a number is taken as a delimiter.

*Stat* nodes are divided into four classes.

|                       |                                                              |
|-----------------------|--------------------------------------------------------------|
| <i>Transformers</i> , | which map input vector elements into output vector elements; |
| <i>Summarizers</i> ,  | which calculate statistics of a vector;                      |
| <i>Translators</i> ,  | which convert among formats; and                             |
| <i>Generators</i> ,   | which are sources of definable vectors.                      |

Below is a list of synopses for *stat* nodes. Most nodes accept options indicated by a leading minus (-). In general, an option is specified by a character followed by a value, such as *c5*. This is interpreted as *c := 5* (*c* is assigned 5). The following keys are used to designate the expected type of the value:

|               |                                                                                          |
|---------------|------------------------------------------------------------------------------------------|
| <i>c</i>      | characters,                                                                              |
| <i>i</i>      | integer,                                                                                 |
| <i>f</i>      | floating point or integer,                                                               |
| <i>file</i>   | file name, and                                                                           |
| <i>string</i> | string of characters, surrounded by quotes to include a <i>Shell</i> argument delimiter. |

Options without keys are flags. All nodes except *generators* accept files as input, hence it is not indicated in the synopses.

**Transformers:**

|              |                                                                                            |
|--------------|--------------------------------------------------------------------------------------------|
| <b>abs</b>   | [ - <i>ci</i> ] — absolute value<br>columns (similarly for - <i>c</i> options that follow) |
| <b>af</b>    | [ - <i>ci t v</i> ] — arithmetic function<br>titled output, verbose                        |
| <b>ceil</b>  | [ - <i>ci</i> ] — round up to next integer                                                 |
| <b>cusum</b> | [ - <i>ci</i> ] — cumulative sum                                                           |
| <b>exp</b>   | [ - <i>ci</i> ] — exponential                                                              |
| <b>floor</b> | [ - <i>ci</i> ] — round down to next integer                                               |
| <b>gamma</b> | [ - <i>ci</i> ] — gamma                                                                    |
| <b>list</b>  | [ - <i>ci dstring</i> ] — list vector elements<br>delimiter(s)                             |



|               |                                                                                                                                                                                                                                           |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>log</b>    | [ -ci bf ] - logarithm<br>base                                                                                                                                                                                                            |
| <b>mod</b>    | [ -ci mf ] - modulus<br>modulus                                                                                                                                                                                                           |
| <b>pair</b>   | [ -ci Ffile xi ] - pair elements<br>File containing base vector, x group size                                                                                                                                                             |
| <b>power</b>  | [ -ci pf ] - raise to a power<br>power                                                                                                                                                                                                    |
| <b>root</b>   | [ -ci rf ] - take a root<br>root                                                                                                                                                                                                          |
| <b>round</b>  | [ -ci pi si ] - round to nearest integer, .5 rounds to 1<br>places after decimal point, significant digits                                                                                                                                |
| <b>siline</b> | [ -ci if nisf ] - generate a line given slope and intercept<br>intercept, number of positive integers, slope                                                                                                                              |
| <b>sin</b>    | [ -ci ] - sine                                                                                                                                                                                                                            |
| <b>subset</b> | [ -af bf ci Ffile li lf nl np pf si ti ] - generate a subset<br>above, below, File with master vector, interval, leave,<br>master contains element numbers to leave, master con-<br>tains element numbers to pick, pick, start, terminate |

*Summarizers:*

|               |                                                                                                                                              |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>bucket</b> | [ -ai ci Ffile hf ii lf ni ] - break into buckets<br>average size, File containing bucket boundaries, high,<br>interval, low, number         |
| <b>cor</b>    | [ -Ffile ] - correlation coefficient<br>File containing base vector                                                                          |
| <b>hilo</b>   | [ -h l o ox oy ] - find high and low values<br>high only, low only, option form, option form with x<br>prepended, option form with y prepend |
| <b>lreg</b>   | [ -Ffile i o s ] - linear regression<br>File containing base vector, intercept only, option form for<br>siline, slope only                   |
| <b>mean</b>   | [ -ff ni pf ] - (trimmed) arithmetic mean<br>fraction, number, percent                                                                       |
| <b>point</b>  | [ -ff ni pf s ] - point from empirical cumulative density<br>function<br>fraction, number, percent, sorted input                             |
| <b>prod</b>   | - internal product                                                                                                                           |
| <b>qsort</b>  | [ -ci ] - quick sort                                                                                                                         |
| <b>rank</b>   | - vector rank                                                                                                                                |
| <b>total</b>  | - sum total                                                                                                                                  |
| <b>var</b>    | - variance                                                                                                                                   |

*Translators:*

|            |                                                                                                                                                                                                                                                              |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>bar</b> | [ -a b f g ri wi xf xa yf ya ylf yhf ] - build a bar chart<br>suppress axes, bold, suppress frame, suppress grid, region,<br>width in percent, x origin, suppress x-axis label, y origin,<br>suppress y-axis label, y-axis lower bound, y-axis high<br>bound |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- hist** [ -a b f g r i x f x a y f y a y l f y h f ] - build a histogram  
 suppress axes, bold, suppress frame, suppress grid, region,  
 x origin, suppress x-axis label, y origin, suppress y-axis  
 label, y-axis lower bound, y-axis high bound
- label** [ -b c Ffile h p r i x x u y y r ] - label the axis of a GPS  
 file  
 bar chart input, retain case, label File, histogram input,  
 plot input, rotation, x-axis, upper x-axis, y-axis, right y-  
 axis
- pie** [ -b o p p n i p p i r i v x i y i ] - build a pie chart  
 bold, values outside pie, value as percentage(:=100), value  
 as percentage(:=i), draw percent of pie, region, no values,  
 x origin, y origin  
 Unlike other nodes, input is lines of the form  
 [ < i e f c c > ] value [label]  
 ignore (don't draw) slice, explode slice, fill slice,  
 color slice c=( black, red, green, blue)
- plot** [ -a b cstring d f Ffile g m r i x f x a x i f x h f x l f x n i x t  
 y f y a y i f y h f y l f y n i y t ] - plot a graph  
 suppress axes, bold, plotting characters, disconnected,  
 suppress frame, File containing x vector, suppress grid,  
 mark points, region, x origin, suppress x-axis label, x  
 interval, x high bound, x low bound, number of ticks on  
 x-axis, suppress x-axis title, y origin, suppress y-axis label,  
 y interval, y high bound, y low bound, number of ticks on  
 y-axis, suppress y-axis title
- title** [ -b c lstring vstring ustring ] - title a vector or a GPS  
 title bold, retain case, lower title, upper title, vector title

**Generators:**

- gas** [ -c i j f n i s f t f ] - generate additive sequence  
 interval, number, start, terminate
- prime** [ -c i h i l i n i ] - generate prime numbers  
 high, low, number
- rand** [ -c i h f l f m f n i s i ] - generate random sequence  
 high, low, multiplier, number, seed

**RESTRICTIONS**

Some nodes have a limit on the size of the input vector.

**SEE ALSO**

graphics(1G), gps(4).

**NAME**

stlogin – sign on to synchronous terminal

**SYNOPSIS**

stlogin [ delay ]

**DESCRIPTION**

The *stlogin* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It is invoked by the system when a synchronous terminal requests service on a connected synchronous line. You can direct your synchronous terminal to request service by first hitting the LOCAL key and then hitting the S/R key.

*Stlogin* asks for your user name and your password. If you have a password, both must be entered before the S/R key is hit. The password field is not displayed on the screen as you enter it.

At some installations, an option may be invoked that will require you to enter a second “external” password. This will occur only for dial-up connections, and will be prompted by the message “External security:”. Both passwords are required for a successful login.

If password aging has been invoked by the super-user on your behalf, your password may have expired. In this case, you will be shunted into *passwd*(1) to change it, after which you may attempt to login again.

If you do not complete the login successfully within the period specified by *delay* (e.g., 60 seconds), you are likely to be silently disconnected.

After a successful login, accounting files are updated, you will be informed of the existence (if any) of mail, and the profiles (i.e., */etc/profile* and *\$HOME/.profile*) (if any) are executed (see *profile*(4)). *Stlogin* initializes the user and group IDs and the working directory, then executes a command interpreter (usually *sh*(1)) according to specifications found in the */etc/passwd* file. Argument 0 of the command interpreter is – followed by the last component of the interpreter’s path name. The *environment* (see *environ*(5)) is initialized to:

```
HOME=your-login-directory
PATH=:/bin:/usr/bin
LOGNAME=your-login-name
```

**FILES**

|                            |                                   |
|----------------------------|-----------------------------------|
| <i>/etc/utmp</i>           | accounting                        |
| <i>/etc/wtmp</i>           | accounting                        |
| <i>/usr/mail/your-name</i> | mailbox for user <i>your-name</i> |
| <i>/etc/motd</i>           | message-of-the-day                |
| <i>/etc/passwd</i>         | password file                     |
| <i>/etc/profile</i>        | system profile                    |
| <i>\$HOME/.profile</i>     | personal profile                  |

**SEE ALSO**

mail(1), newgrp(1), passwd(1), sh(1), su(1), passwd(4), profile(4), environ(5).

**DIAGNOSTICS**

*Login incorrect*

if the user name or the password is incorrect.

*No shell, cannot open password file, no directory:*

consult a UNIX programming counselor.

*Your password has expired. Choose a new one.*

if password aging is implemented.

**NAME**

**strip** — strip symbol and line number information from a common object file

**SYNOPSIS**

**strip** [-l] [-x] [-r] [-s] [-V] file-names

**DESCRIPTION**

The *strip* command strips the symbol table and line number information from common object files, including archives. Once this has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using any of the following options:

- l Strip line number information only; do not strip any symbol table information.
- x Do not strip static or external symbol information.
- r Reset the relocation indexes into the symbol table.
- s Reset the line number indexes into the symbol table (do not remove). reset the relocation indexes into the symbol table.
- V Version of strip command executing.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *strip* will complain and terminate without stripping *file-name* unless the *-r* flag is used.

If the *strip* command is executed on a common archive file (see *ar(4)*) the archive symbol table will be removed. The archive symbol table must be restored by executing the *ar(1)* command with the *s* option before the archive can be link edited by the *ld(1)* command. *Strip(1)* will instruct the user with appropriate warning messages when this situation arises.

The purpose of this command is to reduce the file storage overhead taken by the object file.

**FILES**

/usr/tmp/strip?????

**SEE ALSO**

*as(1)*, *cc(1)*, *ld(1)*, *ar(4)*, *a.out(4)*.

**DIAGNOSTICS**

- strip: name: cannot open  
if *name* cannot be read.
- strip: name: bad magic  
if *name* is not an appropriate common object file.
- strip: name: relocation entries present; cannot strip  
if *name* contains relocation entries and the *-r* flag is not used, the symbol table information cannot be stripped.

**NAME**

*strip* — remove symbols and relocation bits

**SYNOPSIS**

*strip* name ...

**DESCRIPTION**

*Strip* removes the symbol table and relocation bits ordinarily attached to the output of the assembler and link editor. 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)*.

If *name* is an archive file, *strip* will remove the local symbols from any *a.out* format files it finds in the archive. Certain libraries, such as those residing in */lib*, have no need for local symbols. By deleting them, the size of the archive is decreased and link editing performance is increased.

**FILES**

*/tmp/stm\** temporary file

**SEE ALSO**

*ld(1)*, *ar(4)*, *a.out(4)*.

**NAME**

*ststat* — report synchronous terminal facilities status

**SYNOPSIS**

*ststat* [ options ]

**DESCRIPTION**

*Ststat* prints certain information about synchronous terminal facilities. The information that is displayed is controlled by *options*:

- a**            Use all print *options*. (This is shorthand notation for —**g**, —**l**, —**p**, and —**t**.)
- c** *corefile* Use the file *corefile* in place of */dev/kmem*.
- g**            Print information about gen paramaters. (Number of synchronous lines, number of printer ports, number of terminal ports, number of message headers, and sizes of receive and transmit buffer areas.)
- l**            Print information about synchronous lines. (For each synchronous line, whether or not the protocol script is running and whether or not it has established communications with a controller on the line.)
- n** *namelist* The argument will be taken as the name of an alternate *namelist* (*/unix* is the default).
- p**            Print printer port status information. (For each assigned printer port, give the assigned path name and the synchronous line number and device code for the assigned printer.) If none of the print *options* —**a**, —**g**, —**l**, or —**t** are specified, —**p** is supplied as a default.
- t**            Print terminal port status information. (For each active terminal port, give the path name of the terminal device and tell whether an open is waiting to be assigned to a terminal, open to an active terminal, or open to a device that has hung up.)

**FILES**

*/dev*            searched to find terminal (“*tty*”) names  
*/dev/kmem*      memory  
*/unix*           system namelist

**SEE ALSO**

*st(1M)*, *st(7)*.

**DIAGNOSTICS**

Can't read system namelist.

Unable to find system name entries in the *namelist* file.

No synchronous terminal lines in *namelist*.

Synchronous terminals are not configured in the system in the *namelist* file.

Can't open *corefile*.

Unable to open the specified *corefile* file.

Can't read *corefile*.

A read failed on the *corefile* file.

*/dev/?????*

The name of an active terminal port could not be found in the */dev* directory.

**BUGS**

Things can change while *ststat* is running; the picture it gives is only a close approximation to reality.

## NAME

stty — set the options for a terminal

## SYNOPSIS

stty [ -a ] [ -g ] [ options ]

## DESCRIPTION

*Stty* sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the **-a** option, it reports all of the option settings; with the **-g** option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below may be found in *termio(7)* for asynchronous lines, or in *stermio(7)* for synchronous lines in the *UNIX System Administrator's Manual*. Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

## Control Modes

**parenb (-parenb)** enable (disable) parity generation and detection.  
**parodd (-parodd)** select odd (even) parity.  
**cs5 cs6 cs7 cs8** select character size (see *termio(7)*).  
**0** hang up phone line immediately.  
**50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb**  
 Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)  
**hupcl (-hupcl)** hang up (do not hang up) DATA-PHONE® connection on last close.  
**hup (-hup)** same as **hupcl (-hupcl)**.  
**cstopb (-cstopb)** use two (one) stop bits per character.  
**cread (-cread)** enable (disable) the receiver.  
**clocal (-clocal)** assume a line without (with) modem control.

## Input Modes

**ignbrk (-ignbrk)** ignore (do not ignore) break on input.  
**brkint (-brkint)** signal (do not signal) INTR on break.  
**ignpar (-ignpar)** ignore (do not ignore) parity errors.  
**parmrk (-parmrk)** mark (do not mark) parity errors (see *termio(7)*).  
**inpck (-inpck)** enable (disable) input parity checking.  
**istrip (-istrip)** strip (do not strip) input characters to seven bits.  
**inlcr (-inlcr)** map (do not map) NL to CR on input.  
**igncr (-igncr)** ignore (do not ignore) CR on input.  
**icrnl (-icrnl)** map (do not map) CR to NL on input.  
**iucL (-iucL)** map (do not map) upper-case alphabets to lower case on input.  
**ixon (-ixon)** enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.  
**ixany (-ixany)** allow any character (only DC1) to restart output.  
**ixoff (-ixoff)** request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

## Output Modes

**opost (-opost)** post-process output (do not post-process output; ignore all other output modes).  
**olcuc (-olcuc)** map (do not map) lower-case alphabets to upper case on output.

|                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>onlcr</b> ( <b>-onlcr</b> )                   | map (do not map) NL to CR-NL on output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>ocrnl</b> ( <b>-ocrnl</b> )                   | map (do not map) CR to NL on output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>onocr</b> ( <b>-onocr</b> )                   | do not (do) output CRs at column zero.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>onlret</b> ( <b>-onlret</b> )                 | on the terminal NL performs (does not perform) the CR function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>ofill</b> ( <b>-ofill</b> )                   | use fill characters (use timing) for delays.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>ofdel</b> ( <b>-ofdel</b> )                   | fill characters are DELs (NULs).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>cr0 cr1 cr2 cr3</b>                           | select style of delay for carriage returns (see <i>termio</i> (7)).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>nl0 nl1</b>                                   | select style of delay for line-feeds (see <i>termio</i> (7)).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>tab0 tab1 tab2 tab3</b>                       | select style of delay for horizontal tabs (see <i>termio</i> (7) or <i>stermio</i> (7)).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>bs0 bs1</b>                                   | select style of delay for backspaces (see <i>termio</i> (7)).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>ff0 ff1</b>                                   | select style of delay for form-feeds (see <i>termio</i> (7)).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>vt0 vt1</b>                                   | select style of delay for vertical tabs (see <i>termio</i> (7)).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Local Modes</b>                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>isig</b> ( <b>-isig</b> )                     | enable (disable) the checking of characters against the special control characters INTR and QUIT.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>icanon</b> ( <b>-icanon</b> )                 | enable (disable) canonical input (ERASE and KILL processing).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>xcase</b> ( <b>-xcase</b> )                   | canonical (unprocessed) upper/lower-case presentation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>echo</b> ( <b>-echo</b> )                     | echo back (do not echo back) every character typed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>echoe</b> ( <b>-echoe</b> )                   | echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does <i>not</i> keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.                                                                                                                                                                                                                                                                                             |
| <b>echok</b> ( <b>-echok</b> )                   | echo (do not echo) NL after KILL character.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>lfkc</b> ( <b>-lfkc</b> )                     | the same as <b>echok</b> ( <b>-echok</b> ); obsolete.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>echoNL</b> ( <b>-echoNL</b> )                 | echo (do not echo) NL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>noflsh</b> ( <b>-noflsh</b> )                 | disable (enable) flush after INTR or QUIT.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>stwrap</b> ( <b>-stwrap</b> )                 | disable (enable) truncation of lines longer than 79 characters on a synchronous line.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>stflush</b> ( <b>-stflush</b> )               | enable (disable) flush on a synchronous line after every <i>write</i> (2).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>stappl</b> ( <b>-stappl</b> )                 | use application mode (use line mode) on a synchronous line.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Control Assignments</b>                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <i>control-character c</i>                       | set <i>control-character</i> to <i>c</i> , where <i>control-character</i> is <b>erase</b> , <b>kill</b> , <b>intr</b> , <b>quit</b> , <b>eof</b> , <b>eol</b> , <b>ctab</b> , <b>min</b> , or <b>time</b> ( <b>ctab</b> is used with <b>-stappl</b> ; see <i>stermio</i> (7)), ( <b>min</b> and <b>time</b> are used with <b>-icanon</b> ; see <i>termio</i> (7)). If <i>c</i> is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., “^d” is a CTRL-d); “^?” is interpreted as DEL and “^-” is interpreted as undefined. |
| <b>line i</b>                                    | set line discipline to <i>i</i> ( $0 < i < 127$ ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Combination Modes</b>                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>evenp</b> or <b>parity</b>                    | enable <b>parenb</b> and <b>cs7</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>oddp</b>                                      | enable <b>parenb</b> , <b>cs7</b> , and <b>parodd</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>-parity</b> , <b>-evenp</b> , or <b>-oddp</b> | disable <b>parenb</b> , and set <b>cs8</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>raw</b> ( <b>-raw</b> or <b>cooked</b> )      | enable (disable) raw input and output (no ERASE,                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |



**nl** (**-nl**) KILL, INTR, QUIT, EOT, or output post processing).  
unset (set) **icrnl**, **onlcr**. In addition **-nl** unsets  
**inlcr**, **igncr**, **ocrnl**, and **onlret**.

**lcase** (**-lcase**) set (unset) **xcase**, **iucL**, and **olcuc**.

**LCASE** (**-LCASE**) same as **lcase** (**-lcase**).

**tabs** (**-tabs** or **tab3**) preserve (expand to spaces) tabs when printing.

**ek** reset ERASE and KILL characters back to normal #  
and @.

**sane** resets all modes to some reasonable values.

**term** set all modes suitable for the terminal type *term*,  
where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**,  
or **tek**.

**SEE ALSO**

**tabs**(1), **ioctl**(2).  
**sternio**(7), **termio**(7) in the *UNIX System Administrator's Manual*.

**NAME**

*su* — become super-user or another user

**SYNOPSIS**

**su** [ - ] [ name [ arg ... ] ]

**DESCRIPTION**

*Su* allows one to become another user without logging off. The default user *name* is *root* (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless one is already super-user). If the password is correct, *su* will execute a new shell with the user ID set to that of the specified user. To restore normal user ID privileges, type an EOF to the new shell.

Any additional arguments are passed to the shell, permitting the super-user to run shell procedures with restricted privileges (an *arg* of the form *-c string* executes *string* via the shell). When additional arguments are passed, */bin/sh* is always used. When no additional arguments are passed, *su* uses the shell specified in the password file.

An initial *-* flag causes the environment to be changed to the one that would be expected if the user actually logged in again. This is done by invoking the shell with an *arg0* of *-su* causing the *.profile* in the home directory of the new user ID to be executed. Otherwise, the environment is passed along with the possible exception of *\$PATH*, which is set to */bin:/etc:/usr/bin* for *root*. Note that the *.profile* can check *arg0* for *-sh* or *-su* to determine how it was invoked.

**FILES**

|                        |                        |
|------------------------|------------------------|
| <i>/etc/passwd</i>     | system's password file |
| <i>\$HOME/.profile</i> | user's profile         |

**SEE ALSO**

*env*(1), *login*(1), *sh*(1), *environ*(5).

**NAME**

sum - print checksum and block count of a file

**SYNOPSIS**

sum [ -r ] file

**DESCRIPTION**

*Sum* calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option `-r` causes an alternate algorithm to be used in computing the checksum.

**SEE ALSO**

wc(1).

**DIAGNOSTICS**

“Read error” is indistinguishable from end of file on most devices; check the block count.

**NAME**

`sync` — update the super block

**SYNOPSIS**

`sync`

**DESCRIPTION**

*Sync* executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync(2)* for details.

**SEE ALSO**

*sync(2)*.

## NAME

tabs — set tabs on a terminal

## SYNOPSIS

tabs [ tabspec ] [ +mn ] [ -Ttype ]

## DESCRIPTION

*Tabs* sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user must of course be logged in on a terminal with remotely-settable hardware tabs.

Users of GE TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings: the first number in a list of tab settings becomes the *left margin* on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for *tabspec*: "canned," repetitive, arbitrary, and file. If no *tabspec* is given, the default value is -8, i.e., UNIX "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

- code Gives the name of one of a set of "canned" tabs. The legal codes and their meanings are as follows:
- a 1,10,16,36,72  
Assembler, IBM S/370, first format
- a2 1,10,16,40,72  
Assembler, IBM S/370, second format
- c 1,8,12,16,20,55  
COBOL, normal format
- c2 1,6,10,14,49  
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:  
    <:t-c2 m6 s66 d:>
- c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67  
COBOL compact format (columns 1-6 omitted), with more tabs than -c2. This is the recommended format for COBOL. The appropriate format specification is:  
    <:t-c3 m6 s66 d:>
- f 1,7,11,15,19,23  
FORTRAN
- p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61  
PL/I
- s 1,10,55  
SNOBOL
- u 1,12,20,44  
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

- n A repetitive specification requests tabs at columns  $1+n$ ,  $1+2*n$ , etc. Note that such a setting leaves a left margin of  $n$  columns on TermiNet terminals *only*. Of particular importance is the value

-8: this represents the UNIX "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the *nroff* -h option for high-speed output. Another special case is the value -0, implying no tabs at all.

*n1, n2, ...*

The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

--*file* If the name of a file is given, *tabs* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as -8. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:

```
tabs -- file; pr file
```

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

-*Type* *Tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in *term*(5). If no -T flag is supplied, *tabs* searches for the *STERM* value in the *environment* (see *environ*(5)). If no *type* can be found, *tabs* tries a sequence that will work for many terminals.

+*mn* The margin argument may be used for some terminals. It causes all tabs to be moved 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.

Tab and margin setting is performed via the standard output.

#### DIAGNOSTICS

|                          |                                                                                                                                         |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>illegal tabs</i>      | when arbitrary tabs are ordered incorrectly.                                                                                            |
| <i>illegal increment</i> | when a zero or missing increment is found in an arbitrary specification.                                                                |
| <i>unknown tab code</i>  | when a "canned" code cannot be found.                                                                                                   |
| <i>can't open</i>        | if -- <i>file</i> option used, and file can't be opened.                                                                                |
| <i>file indirection</i>  | if -- <i>file</i> option used and the specification in that file points to yet another file. Indirection of this form is not permitted. |

#### SEE ALSO

*nroff*(1), *environ*(5), *term*(5).

#### BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

It is generally impossible to usefully change the left margin without also setting tabs.

*Tabs* clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 40.

**NAME**

**tail** — deliver the last part of a file

**SYNOPSIS**

**tail** [ ±[number][lbcf] ] [ file ]

**DESCRIPTION**

*Tail* copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance +*number* from the beginning, or -*number* from the end of the input (if *number* is null, the value 10 is assumed). *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.

With the -f (“follow”) option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it 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**

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

## NAME

tar — tape file archiver

## SYNOPSIS

tar [ key ] [ files ]

## DESCRIPTION

*Tar* saves and restores files on magnetic tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

- r** The named *files* are written on the end of the tape. The **c** function implies this function.
- x** The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.
- t** The names of the specified files are listed each time that they occur on the tape. If no *files* argument is given, all the names on the tape are listed.
- u** The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape.
- c** Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This command implies the **r** function.

The following characters may be used in addition to the letter that selects the desired function:

- 0,...,7** This modifier selects the drive on which the tape is mounted. The default is 1.
- v** Normally, *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.
- w** causes *tar* to 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 means "no".
- f** causes *tar* to use the next argument as the name of the archive instead of `/dev/mt?`. If the name of the file is `-`, *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 pipeline. *Tar* can also be used to move hierarchies with the command:
 

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```
- b** causes *tar* to use the next argument 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 (see **f** above). The block size is determined automatically when reading tapes (key letters **x** and **t**).
- l** tells *tar* to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.



**m** tells *tar* to not restore the modification times. The modification time of the file will be the time of extraction.

**FILES**

/dev/mt?  
/tmp/tar\*

**DIAGNOSTICS**

Complaints about bad key characters and tape read/write errors.  
Complaints if enough memory is not available to hold the link tables.

**BUGS**

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.  
The **b** option should not be used with archives that are going to be updated.  
The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.  
The current limit on file-name length is 100 characters.

## NAME

*tbl* — format tables for *nroff* or *troff*

## SYNOPSIS

*tbl* [ *-TX* ] [ *files* ]

## DESCRIPTION

*Tbl* is a preprocessor that formats tables for *nroff* or *troff*(1). The input files are copied to the standard output, except for lines between *.TS* and *.TE* command lines, which are assumed to describe tables and are re-formatted by *tbl*. (The *.TS* and *.TE* command lines are not altered by *tbl*).

*.TS* is followed by global options. The available global options are:

|                         |                                                                                        |
|-------------------------|----------------------------------------------------------------------------------------|
| <b>center</b>           | center the table (default is left-adjust);                                             |
| <b>expand</b>           | make the table as wide as the current line length;                                     |
| <b>box</b>              | enclose the table in a box;                                                            |
| <b>doublebox</b>        | enclose the table in a double box;                                                     |
| <b>allbox</b>           | enclose each item of the table in a box;                                               |
| <b>tab</b> ( <i>x</i> ) | use the character <i>x</i> instead of a tab to separate items in a line of input data. |

The global options, if any, are terminated with a semi-colon (;).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except that the last format line (which must end with a period) describes *all* remaining lines of the table. Each column of each line of the table is described by a single key-letter, optionally followed by specifiers that determine the font and point size of the corresponding item, that indicate where vertical bars are to appear between columns, that determine column width, inter-column spacing, etc. The available key-letters are:

|          |                                                                                                                            |
|----------|----------------------------------------------------------------------------------------------------------------------------|
| <b>c</b> | center item within the column;                                                                                             |
| <b>r</b> | right-adjust item within the column;                                                                                       |
| <b>l</b> | left-adjust item within the column;                                                                                        |
| <b>n</b> | numerically adjust item in the column: units positions of numbers are aligned vertically;                                  |
| <b>s</b> | span previous item on the left into this column;                                                                           |
| <b>a</b> | center longest line in this column and then left-adjust all other lines in this column with respect to that centered line; |
| <b>^</b> | span down previous entry in this column;                                                                                   |
| <b>_</b> | replace this entry with a horizontal line;                                                                                 |
| <b>=</b> | replace this entry with a double horizontal line.                                                                          |

The characters **B** and **I** stand for the bold and italic fonts, respectively; the character **|** indicates a vertical line between columns.

The format lines are followed by lines containing the actual data for the table, followed finally by *.TE*. Within such data lines, data items are normally separated by *tab* characters.

If a data line consists of only **\_** or **=**, a single or double line, respectively, is drawn across the table at that point; if a *single item* in a data line consists of only **\_** or **=**, then that item is replaced by a single or double line.

Full details of all these and other features of *tbl* are given in the reference manual cited below.

The *-TX* option forces *tbl* to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., line printers).

If no file names are given as arguments (or if `-` is specified as the last argument), `tbl` reads the standard input, so it may be used as a filter. When it is used with `eqn(1)` or `neqn`, `tbl` should come first to minimize the volume of data passed through pipes.

**EXAMPLE**

If we let `→` represent a tab (which should be typed as a genuine tab), then the input:

```
.TS
center box ;
cB s s
cI | cI s
^ | c c
l | n n .
Household Population

Town→Households
→Number→Size
=
Bedminster→789→3.26
Bernards Twp.→3087→3.74
Bernardsville→2018→3.30
Bound Brook→3425→3.04
Bridgewater→7897→3.81
Far Hills→240→3.19
.TE
```

yields:

| Household Population |            |      |
|----------------------|------------|------|
| Town                 | Households |      |
|                      | Number     | Size |
| Bedminster           | 789        | 3.26 |
| Bernards Twp.        | 3087       | 3.74 |
| Bernardsville        | 2018       | 3.30 |
| Bound Brook          | 3425       | 3.04 |
| Bridgewater          | 7897       | 3.81 |
| Far Hills            | 240        | 3.19 |

**SEE ALSO**

*TBL—A Program to Format Tables in the UNIX System Document Processing Guide.*

`cw(1)`, `eqn(1)`, `mm(1)`, `mmt(1)`, `nroff(1)`, `troff(1)`, `mm(5)`, `mv(5)`.

**BUGS**

See *BUGS* under `nroff(1)`.

**NAME**

`tc` — phototypesetter simulator

**SYNOPSIS**

`tc [ -t ] [ -sn ] [ -pl ] [ file ]`

**DESCRIPTION**

`Tc` interprets its input (standard input default) as device codes for a Wang Laboratories, Inc. C/A/T phototypesetter. The standard output of `tc` is intended for a Tektronix 4014 terminal with ASCII and APL character sets. The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, with overstruck combinations where necessary. Typical usage is:

```
troff -t files | tc
```

At the end of each page, `tc` waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command `e` will *suppress* the screen erase before the next page; `sn` will cause the next `n` pages to be skipped; and `!cmd` will send `cmd` to the shell.

The command line options are:

- t Don't wait between pages (for directing output into a file).
- sn Skip the first `n` pages.
- pl Set page length to `l`; `l` may include the scale factors `p` (points), `i` (inches), `c` (centimeters), and `P` (picas); default is picas.

**SEE ALSO**

4014(1), sh(1), tplot(1G), troff(1).

**BUGS**

Font distinctions are lost.

**NAME**

tee - pipe fitting

**SYNOPSIS**

tee [ -i ] [ -a ] [ file ] ...

**DESCRIPTION**

*Tee* transcribes the standard input to the standard output and makes copies in the *files*. The *-i* option ignores interrupts; the *-a* option causes the output to be appended to the *files* rather than overwriting them.

## NAME

test — condition evaluation command

## SYNOPSIS

```
test expr
[ expr ]
```

## DESCRIPTION

*Test* evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

- r *file* true if *file* exists and is readable.
- w *file* true if *file* exists and is writable.
- x *file* true if *file* exists and is executable.
- f *file* true if *file* exists and is a regular file.
- d *file* true if *file* exists and is a directory.
- c *file* true if *file* exists and is a character special file.
- b *file* true if *file* exists and is a block special file.
- p *file* true if *file* exists and is a named pipe (fifo).
- u *file* true if *file* exists and its set-user-ID bit is set.
- g *file* true if *file* exists and its set-group-ID bit is set.
- k *file* true if *file* exists and its sticky bit is set.
- s *file* true if *file* exists and has a size greater than zero.
- t [ *fildev* ] true if the open file whose file descriptor number is *fildev* (1 by default) is associated with a terminal device.
- z *s1* true if the length of string *s1* is zero.
- n *s1* true if the length of the string *s1* is non-zero.
- s1* = *s2* true if strings *s1* and *s2* are identical.
- s1* != *s2* true if strings *s1* and *s2* are *not* identical.
- s1* true if *s1* is *not* the null string.
- n1* —eq *n2* true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons —ne, —gt, —ge, —lt, and —le may be used in place of —eq.

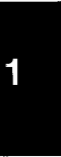
These primaries may be combined with the following operators:

- ! unary negation operator.
- a binary *and* operator.
- o binary *or* operator (—a has higher precedence than —o).
- ( *expr* ) parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

## SEE ALSO

find(1), sh(1).



**WARNING**

In the second form of the command (i.e., 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.

**NAME**

time — time a command

**SYNOPSIS**

time command

**DESCRIPTION**

The *command* is executed; after it is complete, *time* prints 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 execution time can depend on what kind of memory the program happens to land in; the user time in MOS is often half what it is in core.

The times are printed on standard error.

**SEE ALSO**

timex(1), times(2).



**NAME**

`timex` — time a command; report process data and system activity

**SYNOPSIS**

`timex` [options] *command*

**DESCRIPTION**

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on standard error.

*Options* are:

- p List process accounting records for *command* and all its children. Suboptions **f**, **h**, **k**, **m**, **r**, and **t** modify the data items reported, as defined in *acctcom*(1). The number of blocks read or written and the number of characters transferred are always reported.
- o Report the total number of blocks read or written and total characters transferred by *command* and all its children.
- s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

**SEE ALSO**

*acctcom*(1), *sar*(1).

**WARNING**

Process records associated with *command* are selected from the accounting file `/usr/adm/pacct` by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

**EXAMPLES**

A simple example:

```
timex -ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
timex -opskmt sh
      session commands
EOT
```

## NAME

toc — graphical table of contents routines

## SYNOPSIS

**dtoc** [directory]  
**ttoc** mm-file  
**vtoc** [-cdhnmisvn] [TTOC file]

## DESCRIPTION

All of the commands listed below reside in `/usr/bin/graf` (see *graphics(1G)*).

**dtoc** Dtoc makes a textual table of contents, TTOC, of all subdirectories beginning at *directory* (*directory* defaults to `.`). The list has one entry per directory. The entry fields from left to right are level number, directory name, and the number of ordinary readable files in the directory. *Dtoc* is useful in making a visual display of all or parts of a file system. The following will make a visual display of all the readable directories under `/`:

```
dtoc / | vtoc | td
```

**ttoc** Output is the table of contents generated by the `.TC` macro of *mm(1)* translated to TTOC format. The input is assumed to be a *mm* file that uses the `.H` family of macros for section headers. If no *file* is given, the standard input is assumed.

**vtoc** *Vtoc* produces a GPS describing a hierarchy chart from a TTOC. The output drawing consists of boxes containing text connected in a tree structure. If no *file* is given, the standard input is assumed. Each TTOC entry describes one box and has the form:  
*id* [*line-weight,line-style*] "*text*" [*mark*]

where:

*id* is an alternating sequence of numbers and dots. The *id* specifies the position of the entry in the hierarchy. The *id* `0`. is the root of the tree.

*line-weight* is either:  
**n**, normal-weight; or  
**m**, medium-weight; or  
**b**, bold-weight.

*line-style* is either:  
**so**, solid-line;  
**do**, dotted-line;  
**dd**, dot-dash line;  
**da**, dashed-line; or  
**ld**, long-dashed

*text* is a character string surrounded by quotes. The characters between the quotes become the contents of the box. To include a quote within a box it must be escaped (`\`).

*mark* is a character string (surrounded by quotes if it contains spaces), with included dots being escaped. The string is put above the top right corner of the box. To include either a quote or a dot within a *mark* it must be escaped.

Entry example: `1.1 b,da "ABC" DEF`

Entries may span more than one line by escaping the new-line

(\new-line).

Comments are surrounded by the /\*,\*/ pair. They may appear anywhere in a TTOC.

Options:

- c** Use text as entered, (default is all upper case).
- d** Connect the boxes with diagonal lines.
- hn** Horizontal interbox space is  $n\%$  of box width.
- i** Suppress the box *id*.
- m** Suppress the box *mark*.
- s** Do not compact boxes horizontally.
- vn** Vertical interbox space is  $n\%$  of box height.

**SEE ALSO**

graphics(1G), gps(4).

**NAME**

**touch** — update access and modification times of a file

**SYNOPSIS**

**touch** [ **-amc** ] [ **mmddhhmm[yy]** ] files

**DESCRIPTION**

*Touch* causes the access and modification times of each argument to be updated. If no time is specified (see *date(1)*) the current time is used. The **-a** and **-m** options cause *touch* to update only the access or modification times respectively (default is **-am**). The **-c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

**SEE ALSO**

*date(1)*, *utime(2)*.

**NAME**

tplot — graphics filters

**SYNOPSIS**

tplot [ -Tterminal [ -e raster ] ]

**DESCRIPTION**

These commands read plotting instructions (see *plot(4)*) from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter \$TERM (see *environ(5)*) is used. Known *terminals* are:

300 DASI 300.

300S DASI 300s.

450 DASI 450.

4014 Tektronix 4014.

ver Versatec D1200A. This version of *plot* places a scan-converted image in `/usr/tmp/raster$$` and sends the result directly to the plotter device, rather than to the standard output. The `-e` option causes a previously scan-converted file *raster* to be sent to the plotter.

**FILES**`/usr/lib/t300``/usr/lib/t300s``/usr/lib/t450``/usr/lib/t4014``/usr/lib/vplot``/usr/tmp/raster$$`**SEE ALSO**

plot(3X), plot(4), term(5).

**NAME**

tr — translate characters

**SYNOPSIS**

tr [ -cds ] [ string1 [ string2 ] ]

**DESCRIPTION**

*Tr* copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options *-cds* may be used:

- c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.
- d Deletes all input characters in *string1*.
- s Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

- [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*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabets. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
tr -cs "[A-Z][a-z]" "\[012*" <file1 >file2
```

**SEE ALSO**

ed(1), sh(1), ascii(5).

**BUGS**

Won't handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

## NAME

troff — typeset text

## SYNOPSIS

troff [ options ] [ files ]

## DESCRIPTION

*Troff* formats text contained in *files* (standard input by default) for a Wang Laboratories, Inc., C/A/T phototypesetter. Its capabilities are described in the *NROFF/TROFF User's Manual* cited below.

An argument consisting of a minus (-) is taken to be a file name corresponding to the standard input. The *options*, which may appear in any order, but must appear before the *files*, are:

- olist Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. 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. (See *BUGS* below.)
- n*N* Number first generated page *N*.
- s*N* Stop every *N* pages. *Troff* will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.
- ra*N* Set register *a* (which must have a one-character name) to *N*.
- i Read standard input after *files* are exhausted.
- q Invoke the simultaneous input-output mode of the *.rd* request.
- z Print only messages generated by *.tm* (terminal message) requests.
- m*name* Prepend to the input *files* the non-compacted (ASCII text) macro file */usr/lib/tmac/tmac.name*.
- c*name* Prepend to the input *files* the compacted macro files */usr/lib/macros/cmp.[nt].[dt].name* and */usr/lib/macros/ucmp.[nt].name*.
- k*name* Compact the macros used in this invocation of *troff*, placing the output in files *[dt].name* in the current directory (see the May 1979 Addendum to the *NROFF/TROFF User's Manual* for details of compacting macro files).
- t Direct output to the standard output instead of the phototypesetter.
- f Refrain from feeding out paper and stopping phototypesetter at the end of the run.
- w Wait until phototypesetter is available, if it is currently busy.
- b Report whether the phototypesetter is busy or available. No text processing is done.
- a Send a printable ASCII approximation of the results to the standard output.
- p*N* Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.
- g Prepare output for the Murray Hill Computation Center phototypesetter and direct it to the standard output (this option is not usable on most systems). This option is not compatible with the *-s* option; furthermore, when this option is invoked, all *.fp* (font position) requests (if any) in the *troff* input must come before the first break, and no *.tl* requests may come before the first break.
- T*name* Use font-width tables for device *name* (the font tables are found in */usr/lib/font/name/\**). Currently, no *names* are supported.

## FILES

|                      |                                    |
|----------------------|------------------------------------|
| /usr/lib/suftab      | suffix hyphenation tables          |
| /tmp/ta\$#           | temporary file                     |
| /usr/lib/tmac/tmac.* | standard macro files and pointers  |
| /usr/lib/macros/*    | standard macro files               |
| /usr/lib/font/*      | font width tables for <i>troff</i> |

## SEE ALSO

*NROFF/TROFF User's Manual* and *A TROFF Tutorial* in the *UNIX System Document Processing Guide*.

*cw(1)*, *eqn(1)*, *mmt(1)*, *nroff(1)*, *tbl(1)*, *tc(1)*, *mm(5)*, *mv(5)*.

## BUGS

*Troff* believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *troff* generates may be off by one day from your idea of what the date is.

When *troff* is used with the *-olist* option inside a pipeline (e.g., with one or more of *cw(1)*, *eqn(1)*, and *tbl(1)*), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.



## NAME

trouble — log a trouble report

## SYNOPSIS

trouble

## DESCRIPTION

The *trouble* command is a front end for the Piscataway Change Management Tracking System (CMTS). It is used to log trouble reports on, or request enhancements to UNIX. Trouble reports will be forwarded to Piscataway via *uucp*(1C), where they are transformed into Modification Requests (MRs).

The command will prompt for the following mandatory fields:

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name:          | The originator's name (F. M. Last, F. Last, or First Last); (3 to 6 letter ID, if they are in the <b>names</b> file)                                                                                                                                                                                                                                                                                                                                                                  |
| Location:      | The external or internal mailing address                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Phone:         | The telephone number (aaaa, aaa-bbb-cccc, 8aaa-bbbb, or aaa-bbb-cccc xdddd)                                                                                                                                                                                                                                                                                                                                                                                                           |
| Type:          | sw (software), <b>hdw</b> (hardware), <b>doc</b> (documentation), <b>enh</b> (enhancement), <b>unk</b> (unknown)                                                                                                                                                                                                                                                                                                                                                                      |
| System:        | The product under discussion (usually <b>unix</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Machine:       | The CPU on which the trouble was found; <b>na</b> if not applicable                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Release:       | The product release number; <b>na</b> if not applicable                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Severity:      | 1 (out of commission, no circumvention), 2 (severity 1 if not fixed by due date (mo/da/yr)), 3 (needed), 4 (can be deferred)                                                                                                                                                                                                                                                                                                                                                          |
| Date required: | The due date for a severity 2 trouble report                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Trouble Area:  | The command or area in which the trouble was found                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Abstract:      | A one-line summary of the problem                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Description:   | The exact description of the problem; <i>ed</i> (1) is the entry mechanism, so an <b>a</b> (append) must first be typed. Once the description has been entered and edited, a <b>w</b> (write) followed by a <b>q</b> (quit) is required. Since <i>nroff</i> is used to format these reports, all examples can be enclosed within the .ES and .EE formatter macros that are supplied by <i>trouble</i> . In addition, any backslashes should be entered using the <b>\e</b> construct. |

A response of ? will cause the expected format of the response to be displayed.

Unless the description states otherwise, the trouble report may be selected to appear in the *MINI-SYSTEM NEWSLETTER*.

## FILES

|                           |                          |
|---------------------------|--------------------------|
| /usr/lib/trouble/tr.a     | archived trouble reports |
| /usr/lib/trouble/instruct | instructions             |
| /usr/lib/trouble/trsh     | trouble report shell     |
| /usr/lib/trouble/trxmit   | re-transmission shell    |
| /usr/lib/trouble/names    | letter ID data base      |

## SEE ALSO

*uucp*(1C).

**NAME**

true, false — provide truth values

**SYNOPSIS**

**true**

**false**

**DESCRIPTION**

*True* does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true
do
    command
done
```

**SEE ALSO**

*sh*(1).

**DIAGNOSTICS**

*True* has exit status zero, *false* nonzero.

**NAME**

tsort — topological sort

**SYNOPSIS**

tsort [ file ]

**DESCRIPTION**

*Tsort* produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

**SEE ALSO**

lorder(1).

**DIAGNOSTICS**

Odd data: there is an odd number of fields in the input file.

**BUGS**

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

**NAME**

tty — get the terminal's name

**SYNOPSIS**

tty [ -l ] [ -s ]

**DESCRIPTION**

*Tty* prints the path name of the user's terminal. The `-l` option prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line. The `-s` option inhibits printing of the terminal's path name, allowing one to test just the exit code.

**EXIT CODES**

2     if invalid options were specified,  
0     if standard input is a terminal,  
1     otherwise.

**DIAGNOSTICS**

“not on an active synchronous line” if the standard input is not a synchronous terminal and `-l` is specified.

“not a tty” if the standard input is not a terminal and `-s` is not specified.

**NAME**

**umask** -- set file-creation mode mask

**SYNOPSIS**

**umask** [ *ooo* ]

**DESCRIPTION**

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod(2)* and *umask(2)*). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat(2)*). 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**).

If *ooo* is omitted, the current value of the mask is printed.

*Umask* is recognized and executed by the shell.

**SEE ALSO**

*chmod(1)*, *sh(1)*, *chmod(2)*, *creat(2)*, *umask(2)*.

**NAME**

`uname` — print name of current UNIX system

**SYNOPSIS**

`uname [ -snrvma ]`

**DESCRIPTION**

*Uname* prints the current system name of UNIX on the standard output file. It is mainly useful to determine what system one is using. The options cause selected information returned by *uname(2)* to be printed:

- `-s` print the system name (default).
- `-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.
- `-v` print the operating system version.
- `-m` print the machine hardware name.
- `-a` print all the above information.

Arguments not recognized default the command to the `-s` option.

**SEE ALSO**

`uname(2)`.

**NAME**

`unget` — undo a previous `get` of an SCCS file

**SYNOPSIS**

`unget` [-rSID] [-s] [-n] files

**DESCRIPTION**

`Unget` undoes the effect of a `get -e` done prior to creating the intended new delta. If a directory is named, `unget` behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of `-` is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

- rSID      Uniquely identifies which delta is no longer intended. (This would have been specified by `get` as the “new delta”). The use of this keyletter is necessary only if two or more outstanding `gets` for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified `SID` is ambiguous, or if it is necessary and omitted on the command line.
- s         Suppresses the printout, on the standard output, of the intended delta’s `SID`.
- n         Causes the retention of the gotten file which would normally be removed from the current directory.

**SEE ALSO**

`delta(1)`, `get(1)`, `sact(1)`.

**DIAGNOSTICS**

Use `help(1)` for explanations.

**NAME**

**uniq** — report repeated lines in a file

**SYNOPSIS**

**uniq** [ **-udc** [ **+n** ] [ **-n** ] ] [ *input* [ *output* ] ]

**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. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **-u** flag is used, just the lines that are not repeated in the original file are output. The **-d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **-u** and **-d** mode outputs.

The **-c** option supersedes **-u** and **-d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

- 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 tabs and spaces from its neighbors.
- +n**     The first *n* characters are ignored. Fields are skipped before characters.

**SEE ALSO**

*comm*(1), *sort*(1).



**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. Compound names are run together, (e.g. **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

**FILES**

```
/usr/lib/unittab
```

**NAME**

uucp, uulog, uuname — unix to unix copy

**SYNOPSIS**

**uucp** [ options ] source-files destination-file

**uulog** [ options ]

**uuname** [ -l ]

**DESCRIPTION****Uucp.**

*Uucp* copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names which *uucp* knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, and only to a destination in PUBDIR (see below). Care should be taken to insure that intermediate nodes in the route are willing to forward information.

The shell metacharacters *?*, *\** and *[...]* appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

- (1) a full path name;
- (2) a path name preceded by *~user* where *user* is a login name on the specified system and is replaced by that user's login directory;
- (3) a path name preceded by *~/user* where *user* is a login name on the specified system and is replaced by that user's directory under PUBDIR;
- (4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

*Uucp* preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod(2)*).

The following options are interpreted by *uucp*:

- d Make all necessary directories for the file copy (default).
- f Do not make intermediate directories for the file copy.
- c Use the source file when copying out rather than copying the file to the spool directory (default).
- C Copy the source file to the spool directory.
- mfile Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.
- nuser Notify *user* on the remote system that a file was sent.
- esys Send the *uucp* command to system *sys* to be executed there. (Note: this will only be successful if the remote machine allows

the *uucp* command to be executed by */usr/lib/uucp/uuxqt.*)

*Uucp* returns on the standard output a string which is the job number of the request. This job number can be used by *uustat* to obtain status or terminate the job.

### Uulog.

*Uulog* queries a summary log of *uucp* and *uux*(1C) transactions in the file */usr/spool/uucp/LOGFILE*.

The options cause *uulog* to print logging information:

- ssys* Print information about work involving system *sys*.
- user* Print information about work done for the specified *user*.

### Uuname.

*Uuname* lists the *uucp* names of known systems. The *-l* option returns the local system name.

### FILES

|                              |                                                     |
|------------------------------|-----------------------------------------------------|
| <i>/usr/spool/uucp</i>       | spool directory                                     |
| <i>/usr/spool/uucppublic</i> | public directory for receiving and sending (PUBDIR) |
| <i>/usr/lib/uucp/*</i>       | other data and program files                        |

### SEE ALSO

*mail*(1), *uux*(1C).

### WARNING

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 *~nuucp* or just *~*).

### BUGS

All files received by *uucp* will be owned by *uucp*.  
The *-m* option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters *? \* [...]* will not activate the *-m* option.

**NAME**

**uustat** — uucp status inquiry and job control

**SYNOPSIS**

**uustat** [ options ]

**DESCRIPTION**

*Uustat* will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. The following *options* are recognized:

- j**jobn* Report the status of the *uucp* request *jobn*. If **all** is used for *jobn*, the status of all *uucp* requests is reported. If *jobn* is omitted, the status of the current user's *uucp* requests is reported.
- k**jobn* Kill the *uucp* request whose job number is *jobn*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.
- r**jobn* Rejuvenate *jobn*. That is *jobn* is touched so that its modification time is set to the current time. This prevents *uuclean* from deleting the job until the jobs modification time reaches the limit imposed by *uuclean*.
- c**hour* Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the super-user.
- u**user* Report the status of all *uucp* requests issued by *user*.
- ss**sys* Report the status of all *uucp* requests which communicate with remote system *sys*.
- o**hour* Report the status of all *uucp* requests which are older than *hour* hours.
- y**hour* Report the status of all *uucp* requests which are younger than *hour* hours.
- mm**mch* Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local *uucp* are provided.
- M**mch* This is the same as the *-m* option except that two times are printed. The time that the last status was obtained and the time that the last successful transfer to that system occurred.
- O* Report the *uucp* status using the octal status codes listed below. If this option is not specified, the verbose description is printed with each *uucp* request.
- q* List the number of jobs and other control files queued for each machine and the time of the oldest and youngest file queued for each machine. If a lock file exists for that system, its date of creation is listed.

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user. Note that only one of the options *-j*, *-m*, *-k*, *-c*, *-r*, can be used with the rest of the other options.

For example, the command:

```
uustat -uhdc -smhtsa -y72
```

will print the status of all *uucp* requests that were issued by user *hdc* to communicate with system *mhtsa* within the last 72 hours. The meanings of the job request status are:

job-number user remote-system command-time status-time status

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

| OCTAL  | STATUS                                               |
|--------|------------------------------------------------------|
| 000001 | the copy failed, but the reason cannot be determined |
| 000002 | permission to access local file is denied            |
| 000004 | permission to access remote file is denied           |
| 000010 | bad <i>uucp</i> command is generated                 |
| 000020 | remote system cannot create temporary file           |
| 000040 | cannot copy to remote directory                      |
| 000100 | cannot copy to local directory                       |
| 000200 | local system cannot create temporary file            |
| 000400 | cannot execute <i>uucp</i>                           |
| 001000 | copy (partially) succeeded                           |
| 002000 | copy finished, job deleted                           |
| 004000 | job is queued                                        |
| 010000 | job killed (incomplete)                              |
| 020000 | job killed (complete)                                |

The meanings of the machine accessibility status are:

system-name time status

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

#### FILES

|                      |                     |
|----------------------|---------------------|
| /usr/spool/uucp      | spool directory     |
| /usr/lib/uucp/L_stat | system status file  |
| /usr/lib/uucp/R_stat | request status file |

#### SEE ALSO

*uucp*(1C).

**NAME**

uuto, uupick — public UNIX-to-UNIX file copy

**SYNOPSIS**

**uuto** [ options ] source-files destination  
**uupick** [ -s system ]

**DESCRIPTION**

*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!user

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *Logname* is the login name of someone on the specified system.

Two *options* are available:

- p Copy the source file into the spool directory before transmission.
- m Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. Specifically the files are sent to

PUBDIR/receive/user/mysystem/files.

The destined recipient is notified by *mail*(1) of the arrival of files.

*Uupick* accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR 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:

- <new-line> Go on to next entry.
- d Delete the entry.
- m [ dir ] Move the entry to named directory *dir* (current directory is default).
- a [ dir ] Same as m except moving all the files sent from *system*.
- p Print the content of the file.
- q Stop.
- EOT (control-d) Same as q.
- !command Escape to the shell to do *command*.
- \* Print a command summary.

*Uupick* invoked with the *-ssystem* option will only search the PUBDIR for files sent from *system*.

**FILES**

PUBDIR/usr/spool/uucppublic public directory

**SEE ALSO**

mail(1), uuclean(1M), uucp(1C), uustat(1C), uux(1C).

## NAME

`uux` - unix to unix command execution

## SYNOPSIS

`uux` [ options ] command-string

## DESCRIPTION

*Uux* will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail(1)*) via *uux*.

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.

File names may be one of

- (1) a full path name;
- (2) a path name preceded by `~xxx` where *xxx* is a login name on the specified system and is replaced by that user's login directory;
- (3) anything else is prefixed by the current directory.

As an example, the command

```
uux "!diff usg!/usr/dan/fl pwba!/a4/dan/fl > !fl.diff"
```

will get the *fl* files from the "usg" and "pwba" machines, execute a *diff* command and put the results in *fl.diff* in the local directory.

Any special shell characters such as `<>|` should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

*Uux* will attempt to get all files to the execution system. For files which are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!uucp b!/usr/file \(c!/usr/file\)
```

will send a *uucp* command to system "a" to get */usr/file* from system "b" and send it to system "c".

*Uux* will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine.

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

- The standard input to *uux* is made the standard input to the *command-string*.
- n Send no notification to user.
- mfile Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

*Uux* returns an ASCII string on the standard output which is the job number. This job number can be used by *uustat* to obtain the status or terminate a job.

## FILES

|                            |                         |
|----------------------------|-------------------------|
| <i>/usr/lib/uucp/spool</i> | spool directory         |
| <i>/usr/lib/uucp/*</i>     | other data and programs |

**SEE ALSO**

uuclean(1M), uucp(1C).

**BUGS**

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.



**NAME**

*val* - validate SCCS file

**SYNOPSIS**

*val* -  
*val* [-s] [-rSID] [-mname] [-ytype] files

**DESCRIPTION**

*Val* determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a -, and named files.

*Val* has a special argument, -, which causes reading of the standard input until an 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 keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

- |        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -s     | The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.                                                                                                                                                                                                                                                                  |
| -rSID  | The argument value <i>SID</i> (SCCS IDentification String) is an SCCS delta number. A check is made to determine if the <i>SID</i> is ambiguous (e. g., r1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (e. g., r1.0 or r1.1.0 are invalid because neither case can exist as a valid delta number). If the <i>SID</i> is valid and not ambiguous, a check is made to determine if it actually exists. |
| -mname | The argument value <i>name</i> is compared with the SCCS %M% keyword in <i>file</i> .                                                                                                                                                                                                                                                                                                                                                                             |
| -ytype | The argument value <i>type</i> is compared with the SCCS %Y% keyword in <i>file</i> .                                                                                                                                                                                                                                                                                                                                                                             |

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

- bit 0 = missing file argument;
- bit 1 = unknown or duplicate keyletter argument;
- bit 2 = corrupted SCCS file;
- bit 3 = can't open file or file not SCCS;
- bit 4 = *SID* is invalid or ambiguous;
- bit 5 = *SID* does not exist;
- bit 6 = %Y%, -y mismatch;
- bit 7 = %M%, -m mismatch;

Note that *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 - a logical OR of the codes generated for each command line and file processed.

**SEE ALSO**

admin(1), delta(1), get(1), prs(1).

**DIAGNOSTICS**

Use *help*(1) for explanations.

**BUGS**

*Val* can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

**NAME**

vc — version control

**SYNOPSIS**

vc [-a] [-t] [-cchar] [-s] [keyword=value ... keyword=value]

**DESCRIPTION**

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the *-t* keyletter (see below). The default control character is colon (:), except as modified by the *-c* keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumeric; the first must be alphabetic. A value is any ASCII string that can be created with *ed(1)*; a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The *-a* keyletter (see below) 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 \.

**Keyletter arguments**

- a* Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.
- t* All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.
- cchar* Specifies a control character to be used in place of :.
- s* Silences warning messages (not error) that are normally printed on the diagnostic output.

**Version Control Statements**

**:dcl keyword[, ..., keyword]**

Used to declare keywords. All keywords must be declared.

**:asg keyword=value**

Used to assign values to keywords. An *asg* statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous *asg*'s for that keyword. Keywords declared, but not assigned values have null values.

**:if condition**

⋮

**:end**

Used to skip lines of the standard input. If the condition is true all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.

The syntax of a condition is:

```

<cond>      ::= [ "not" ] <or>
<or>        ::= <and> | <and> "!" <or>
<and>       ::= <exp> | <exp> "&" <and>
<exp>       ::= "(" <or> ")" | <value> <op> <value>
<op>        ::= "=" | "!=" | "<" | ">"
<value>     ::= <arbitrary ASCII string> | <numeric string>
  
```

The available operators and their meanings are:

|     |                                                                                                              |
|-----|--------------------------------------------------------------------------------------------------------------|
| =   | equal                                                                                                        |
| !=  | not equal                                                                                                    |
| &   | and                                                                                                          |
|     | or                                                                                                           |
| >   | greater than                                                                                                 |
| <   | less than                                                                                                    |
| ( ) | used for logical groupings                                                                                   |
| not | may only occur immediately after the <i>if</i> , and when present, inverts the value of the entire condition |

The > and < operate only on unsigned integer values (e. g.: 012 > 12 is false). All other operators take strings as arguments (e. g.: 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

```

= != > <    all of equal precedence
&
|
  
```

Parentheses may be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

**::text**

Used for keyword replacement on lines that are copied to the standard output. 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. This action is independent of the -a keyletter.

**:on**

**:off**

Turn on or off keyword replacement on all lines.

**:ctl char**

Change the control character to char.

**:msg message**

Prints the given message on the diagnostic output.

**:err message**

Prints 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.

**DIAGNOSTICS**

Use *help(1)* for explanations.

**EXIT CODES**

- 0 — normal
- 1 — any error

**NAME**

vpr — Versatec printer spooler

**SYNOPSIS**

vpr [ options ] [ files ]

**DESCRIPTION**

*Vpr* causes the named *files* to be queued for printing on a Versatec printer. If no names appear, the standard input is assumed; thus *vpr* may be used as a filter.

The following options may be given (each as a separate argument and in any order) before any file name arguments:

- c Make a copy of the file to be sent before returning to the user.
- r Remove the file after sending it.
- m When printing is complete, report that fact by *mail*(1).
- n Do not report the completion of printing by *mail*(1). This is the default option.
- file* Use *file* as a dummy file name to report back in the mail. (This is useful for distinguishing multiple runs, especially when *vpr* is being used as a filter).
- p [ -e *raster* ]  
Use the plot filter *vplot* to output files produced by *graph*(1G). The -e option will cause a previously scan converted file *raster* to be sent to the Versatec.

**EXAMPLES**

Two common uses are:

```
pr [ options ] file | vpr
```

and

```
graph [ options ] file | vpr -p
```

**FILES**

|                  |                                           |
|------------------|-------------------------------------------|
| /etc/passwd      | user's identification and accounting data |
| /usr/spool/vpd/* | spool area                                |
| /usr/lib/vpd     | line printer daemon                       |
| /usr/lib/vpd.pr  | print filter                              |
| /usr/lib/vplot   | plot filter                               |

**SEE ALSO**

dpr(1C), lpr(1), tplot(1G).

**NAME**

wait — await completion of process

**SYNOPSIS**

**wait**

**DESCRIPTION**

Wait until all processes started with **&** have completed, and report on abnormal terminations.

Because the *wait(2)* system call must be executed in the parent process, the shell itself executes *wait*, without creating a new process.

**SEE ALSO**

sh(1).

**BUGS**

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus can't be waited for.

**NAME**

`wc` — word count

**SYNOPSIS**

`wc [ -lwc ] [ names ]`

**DESCRIPTION**

*Wc* counts lines, words and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **-lwc**.

When *names* are specified on the command line, they will be printed along with the counts.



**NAME**

*what* — identify SCCS files

**SYNOPSIS**

*what* files

**DESCRIPTION**

*What* searches the given files for all occurrences of the pattern that *get(1)* substitutes for %Z% (this is *@(#)* at this printing) and prints out what follows until the first *"*, *>*, new-line, *\*, or null character. For example, if the C program in file *f.c* contains

```
char ident[] = "@(#)identification information";
```

and *f.c* is compiled to yield *f.o* and *a.out*, then the command

```
what f.c f.o a.out
```

will print

```
f.c:
      identification information
```

```
f.o:
      identification information
```

```
a.out:
      identification information
```

*What* is intended to be used in conjunction with the command *get(1)*, which automatically inserts identifying information, but it can also be used where the information is inserted manually.

**SEE ALSO**

*get(1)*, *help(1)*.

**DIAGNOSTICS**

Use *help(1)* for explanations.

**BUGS**

It's possible that an unintended occurrence of the pattern *@(#)* could be found just by chance, but this causes no harm in nearly all cases.

## NAME

who — who is on the system

## SYNOPSIS

who [-uTlpdbrtas] [file]

who am i

## DESCRIPTION

*Who* can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX user. It examines the */etc/utmp* file to obtain its information. If *file* is given, that file is examined. Usually, *file* will be */etc/wtmp*, which contains a history of all the logins since the file was last created.

*Who* with the *am i* option identifies the invoking user.

Except for the default *-s* option, the general format for output entries is:

```
name [state] line time activity pid [comment] [exit]
```

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

- u This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory */dev*. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in */etc/inittab* (see *inittab(4)*). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.
- T This option is the same as the *-u* option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a - appears if it is not. *Root* can write to all lines having a + or a - in the *state* field. If a bad line is encountered, a ? is printed.
- l This option lists only those lines on which the system is waiting for someone to login. The *name* field is *LOGIN* in such cases. Other fields are the same as for user entries except that the *state* field doesn't exist.
- p This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in */etc/inittab*. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from */etc/inittab* that spawned this process. See *inittab(4)*.
- d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait(2)*), of the

dead process. This can be useful in determining why a process terminated.

- b This option indicates the time and date of the last reboot.
- r This option indicates the current *run-level* of the *init* process.
- t This option indicates the last change to the system clock (via the *date*(1) command) by *root*. See *su*(1).
- a This option processes */etc/utmp* or the named *file* with all options turned on.
- s This option is the default and lists only the *name*, *line* and *time* fields.

#### FILES

*/etc/utmp*  
*/etc/wtmp*  
*/etc/inittab*

#### SEE ALSO

*init*(1M) in the *UNIX System Administrator's Manual*.  
*date*(1), *login*(1), *mesg*(1), *su*(1), *wait*(2), *inittab*(4), *utmp*(4).

**NAME**

write - write to another user

**SYNOPSIS**

write user [ line ]

**DESCRIPTION**

*Write* copies lines from your terminal to that of another user. When first called, it sends the message:

**Message from yourname (tty??) [ date ]...**

to the person you want to talk to. When it has successfully completed the connection it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point *write* writes EOT on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., tty00); otherwise, the first instance of the user found in */etc/utmp* is assumed and the following message posted:

*user* is logged on more than one place.  
You are connected to "*terminal*".  
Other locations are:  
*terminal*

Permission to write may be denied or granted by use of the *mesg(1)* command. Writing to others is normally allowed by default. Certain commands, in particular *nroff(1)* and *pr(1)* disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write inhibited terminal.

If the character ! is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., (o) for "over") so that the other person knows when to reply. The signal (oo) (for "over and out") is suggested when conversation is to be terminated.

**FILES**

*/etc/utmp* to find user  
*/bin/sh* to execute !

**SEE ALSO**

mail(1), mesg(1), nroff(1), pr(1), sh(1), who(1).

**DIAGNOSTICS**

"*user not logged in*" if the person you are trying to *write* to is not logged in.

## NAME

*xargs* — construct argument list(s) and execute command

## SYNOPSIS

*xargs* [ flags ] [ command [ initial-arguments ] ]

## DESCRIPTION

*Xargs* combines the fixed *initial-arguments* with arguments read from 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 flags specified.

*Command*, which may be a shell file, is searched for, using one's \$PATH. If *command* is omitted, /bin/echo is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if 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 next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see -i flag). Flags -i, -l, and -n determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., -l vs. -n), the last flag has precedence. *Flag* values are:

- i *number*                    *Command* is executed for each non-empty *number* lines of arguments from 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 new-line unless the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted 1 is assumed. Option -x is forced.
- i *replstr*                Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, 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*. Blanks and tabs 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.
- 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.

- t** Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.
- 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 carriage return, skips that particular invocation of *command*.
- x** Causes *xargs* to terminate 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.
- 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 that the character count for *size* includes one extra character for each argument and the count of characters in the command name.
- e***eofstr* *Eofstr* is taken as the logical end-of-file string. Underbar (*\_*) is assumed for the logical EOF string if **-e** is not coded. **-e** with no *eofstr* coded turns off the logical EOF string capability (underbar is taken literally). *Xargs* reads standard input until either end-of-file or the logical EOF string is encountered.

*Xargs* will terminate if either it receives a return code of **-1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh*(1)) with an appropriate value to avoid accidentally returning with **-1**.

#### 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 (2.) many at a time.

1. `ls | xargs -p -l ar r arch`
2. `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
```

#### DIAGNOSTICS

Self explanatory.

**NAME**

yacc — yet another compiler-compiler

**SYNOPSIS**

yacc [ *-vdl* ] 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; specified precedence rules are used to break ambiguities.

The output file, *y.tab.c*, must be compiled by the C compiler to produce a program *yparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex(1)* is useful for creating lexical analyzers usable by *yacc*.

If the *-v* flag is given, the file *y.output* is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the *-d* flag is used, the file *y.tab.h* is generated with the *#define* statements that associate the *yacc*-assigned “token codes” with the user-declared “token names”. This allows source files other than *y.tab.c* to access the token codes.

If the *-l* flag is given, the code produced in *y.tab.c* will *not* contain any *#line* constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in *y.tab.c* under conditional compilation control. By default, this code is not included when *y.tab.c* is compiled. However, when *yacc*'s *-t* option is used, this debugging code will be compiled by default. Independent of whether the *-t* option was used, the runtime debugging code is under the control of *YYDEBUG*, a pre-processor symbol. If *YYDEBUG* has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

**FILES**

|                                      |                                 |
|--------------------------------------|---------------------------------|
| <i>y.output</i>                      |                                 |
| <i>y.tab.c</i>                       |                                 |
| <i>y.tab.h</i>                       | defines for token names         |
| <i>yacc.tmp</i> ,                    |                                 |
| <i>yacc.debug</i> , <i>yacc.acts</i> | temporary files                 |
| <i>/usr/lib/yaccpar</i>              | parser prototype for C programs |

**SEE ALSO**

*lex(1)*.  
*YACC—Yet Another Compiler Compiler* in the *UNIX System Support Tools Guide*.

**DIAGNOSTICS**

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the *y.output* file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

**BUGS**

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

**NAME**

intro — introduction to system calls and error numbers

**SYNOPSIS**

```
#include <errno.h>
```

**DESCRIPTION**

This section describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value. This is almost always `-1`; the individual descriptions specify the details. An error number is also made available in the external variable `errno`. `Errno` is not cleared on successful calls, so it should be tested only after an error has been indicated.

All of the possible error numbers are not listed in each system call description because many errors are possible for most of the calls. The following is a complete list of the error numbers and their names as defined in `<errno.h>`.

**1 EPERM Not owner**

Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

**2 ENOENT No such file or directory**

This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

**3 ESRCH No such process**

No process can be found corresponding to that specified by `pid` in `kill` or `ptrace`.

**4 EINTR Interrupted system call**

An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

**5 EIO I/O error**

Some physical I/O error. This error may in some cases occur on a call following the one to which it actually applies.

**6 ENXIO No such device or address**

I/O on a special file refers to a subdevice which does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not on-line or no disk pack is loaded on a drive.

**7 E2BIG Arg list too long**

An argument list longer than 5,120 bytes is presented to a member of the `exec` family.

**8 ENOEXEC Exec format error**

A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number (see `a.out(4)`).

**9 EBADF Bad file number**

Either a file descriptor refers to no open file, or a read (respectively write) request is made to a file which is open only for writing (respectively reading).



- 10 ECHILD No child processes  
A *wait*, was executed by a process that had no existing or unwaited-for child processes.
- 11 EAGAIN No more processes  
A *fork*, failed because the system's process table is full or the user is not allowed to create any more processes.
- 12 ENOMEM Not enough space  
During an *exec*, *brk*, or *sbrk*, a program asks for more space than the system is able to supply. This is not a temporary condition; the maximum space size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers, or if there is not enough swap space during a *fork*.
- 13 EACCES Permission denied  
An attempt was made to access a file in a way forbidden by the protection system.
- 14 EFAULT Bad address  
The system encountered a hardware fault in attempting to use an argument of a system call.
- 15 ENOTBLK Block device required  
A non-block file was mentioned where a block device was required, e.g., in *mount*.
- 16 EBUSY Mount device busy  
An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment). It will also occur if an attempt is made to enable accounting when it is already enabled.
- 17 EEXIST File exists  
An existing file was mentioned in an inappropriate context, e.g., *link*.
- 18 EXDEV Cross-device link  
A link to a file on another device was attempted.
- 19 ENODEV No such device  
An attempt was made to apply an inappropriate system call to a device; e.g., read a write-only device.
- 20 ENOTDIR Not a directory  
A non-directory was specified where a directory is required, for example in a path prefix or as an argument to *chdir*(2).
- 21 EISDIR Is a directory  
An attempt to write on a directory.
- 22 EINVAL Invalid argument  
Some invalid argument (e.g., dismounting a non-mounted device; mentioning an undefined signal in *signal*, or *kill*; reading or writing a file for which *lseek* has generated a negative pointer). Also set by the math functions described in the (3M) entries of this manual.
- 23 ENFILE File table overflow  
The system's table of open files is full, and temporarily no more *opens* can be accepted.
- 24 EMFILE Too many open files  
No process may have more than 20 file descriptors open at a time.

- 25 ENOTTY Not a typewriter
- 26 ETXTBSY Text file busy  
An attempt to execute a pure-procedure program which is currently open for writing (or reading). Also an attempt to open for writing a pure-procedure program that is being executed.
- 27 EFBIG File too large  
The size of a file exceeded the maximum file size (1,082,201,088 bytes) or ULIMIT; see *ulimit*(2).
- 28 ENOSPC No space left on device  
During a *write* to an ordinary file, there is no free space left on the device.
- 29 EPIPE Illegal seek  
An *lseek* was issued to a pipe.
- 30 EROFS Read-only file system  
An attempt to modify a file or directory was made on a device mounted read-only.
- 31 EMLINK Too many links  
An attempt to make more than the maximum number of links (1000) to a file.
- 32 EPIPE Broken pipe  
A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.
- 33 EDOM Math argument  
The argument of a function in the math package (3M) is out of the domain of the function.
- 34 ERANGE Result too large  
The value of a function in the math package (3M) is not representable within machine precision.
- 35 ENOMSG No message of desired type  
An attempt was made to receive a message of a type that does not exist on the specified message queue; see *msgop*(2).
- 36 EIDRM Identifier Removed  
This error is returned to processes that resume execution due to the removal of an identifier from the file system's name space (see *msgctl*(2), *semctl*(2), and *shmctl*(2)).

## DEFINITIONS

### Process ID

Each active process in the system is uniquely identified by a positive integer called a process ID. The range of this ID is from 0 to 30,000.

### Parent Process ID

A new process is created by a currently active process; see *fork*(2). The parent process ID of a process is the process ID of its creator.

### Process Group ID

Each active process is a member of a process group that is identified by a positive integer called the process group ID. This ID is the process ID of the group leader. This grouping permits the signaling of related processes; see *kill*(2).

### Tty Group ID

Each active process can be a member of a terminal group that is identified

by a positive integer called the *tty* group ID. This grouping is used to terminate a group of related process upon termination of one of the processes in the group; see *exit*(2) and *signal*(2).

### Real User ID and Real Group ID

Each user allowed on the system is identified by a positive integer called a real user ID.

Each user is also a member of a group. The group is identified by a positive integer called the real group ID.

An active process has a real user ID and real group ID that are set to the real user ID and real group ID, respectively, of the user responsible for the creation of the process.

### Effective User ID and Effective Group ID

An active process has an effective user ID and an effective group ID that are used to determine file access permissions (see below). The effective user ID and effective group ID are equal to the process's real user ID and real group ID respectively, unless the process or one of its ancestors evolved from a file that had the set-user-ID bit or set-group ID bit set; see *exec*(2).

### Super-user

A process is recognized as a *super-user* process and is granted special privileges if its effective user ID is 0.

### Special Processes

The processes with a process ID of 0 and a process ID of 1 are special processes and are referred to as *proc0* and *proc1*.

*Proc0* is the scheduler. *Proc1* is the initialization process (*init*). *Proc1* is the ancestor of every other process in the system and is used to control the process structure.

### File Name.

Names consisting of 1 to 14 characters may be used to name an ordinary file, special file or directory.

These characters may be selected from the set of all character values excluding \0 (null) and the ASCII code for / (slash).

Note that it is generally unwise to use \*, ?, [, or ] as part of file names because of the special meaning attached to these characters by the shell. See *sh*(1). Although permitted, it is advisable to avoid the use of unprintable characters in file names.

### Path Name and Path Prefix

A path name is a null-terminated character string starting with an optional slash (/), followed by zero or more directory names separated by slashes, optionally followed by a file name.

More precisely, a path name is a null-terminated character string constructed as follows:

```
<path-name> ::= <file-name> | <path-prefix> <file-name> |
<path-prefix> ::= <rtprefix> | / <rtprefix>
<rtprefix> ::= <dirname> | / <rtprefix> <dirname> /
```

where <file-name> is a string of 1 to 14 characters other than the ASCII slash and null, and <dirname> is a string of 1 to 14 characters (other than the ASCII slash and null) that names a directory.

If a path name begins with a slash, the path search begins at the *root* directory. Otherwise, the search begins from the current working directory.

A slash by itself names the root directory.

Unless specifically stated otherwise, the null path name is treated as if it named a non-existent file.

### Directory.

Directory entries are called links. By convention, a directory contains at least two links, . and .., referred to as *dot* and *dot-dot* respectively. Dot refers to the directory itself and dot-dot refers to its parent directory.

### Root Directory and Current Working Directory.

Each process has associated with it a concept of a root directory and a current working directory for the purpose of resolving path name searches. A process's root directory need not be the root directory of the root file system.

### File Access Permissions.

Read, write, and execute/search permissions on a file are granted to a process if one or more of the following are true:

The process's effective user ID is super-user.

The process's effective user ID matches the user ID of the owner of the file and the appropriate access bit of the "owner" portion (0700) of the file mode is set.

The process's effective user ID does not match the user ID of the owner of the file, and the process's effective group ID matches the group of the file and the appropriate access bit of the "group" portion (070) of the file mode is set.

The process's effective user ID does not match the user ID of the owner of the file, and the process's effective group ID does not match the group ID of the file, and the appropriate access bit of the "other" portion (07) of the file mode is set.

Otherwise, the corresponding permissions are denied.

### Message Queue Identifier

A message queue identifier (msqid) is a unique positive integer created by a *msgget*(2) system call. Each msqid has a message queue and a data structure associated with it. The data structure is referred to as *msqid\_ds* and contains the following members:

```

struct ipc_perm msg_perm; /* operation permission struct */
ushort msg_qnum;          /* number of msgs on q */
ushort msg_qbytes;        /* max number of bytes on q */
ushort msg_lspid;         /* pid of last msgsnd operation */
ushort msg_lrpid;         /* pid of last msgrcv operation */
time_t msg_stime;         /* last msgsnd time */
time_t msg_rtime;         /* last msgrcv time */
time_t msg_ctime;         /* last change time */
                          /* Times measured in secs since */
                          /* 00:00:00 GMT, Jan. 1, 1970 */

```

*Msg\_perm* is a *ipc\_perm* structure that specifies the message operation permission (see below). This structure includes the following members:

```

ushort cuid;              /* creator user id */
ushort cgid;              /* creator group id */
ushort uid;                /* user id */
ushort gid;                /* group id */
ushort mode;              /* r/w permission */

```

**Msg\_qnum** is the number of messages currently on the queue. **Msg\_qbytes** is the maximum number of bytes allowed on the queue. **Msg\_lspid** is the process id of the last process that performed a *msgsnd* operation. **Msg\_lrpid** is the process id of the last process that performed a *msgrcv* operation. **Msg\_stime** is the time of the last *msgsnd* operation, **msg\_rtime** is the time of the last *msgrcv* operation, and **msg\_ctime** is the time of the last *msgctl*(2) operation that changed a member of the above structure.

### Message Operation Permissions.

In the *msgop*(2) and *msgctl*(2) system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

|       |                       |
|-------|-----------------------|
| 00400 | Read by user          |
| 00200 | Write by user         |
| 00060 | Read, Write by group  |
| 00006 | Read, Write by others |

Read and Write permissions on a *msgid* are granted to a process if one or more of the following are true:

The process's effective user ID is super-user.

The process's effective user ID matches **msg\_perm.[c]uid** in the data structure associated with *msgid* and the appropriate bit of the "user" portion (0600) of **msg\_perm.mode** is set.

The process's effective user ID does not match **msg\_perm.[c]uid** and the process's effective group ID matches **msg\_perm.[c]gid** and the appropriate bit of the "group" portion (060) of **msg\_perm.mode** is set.

The process's effective user ID does not match **msg\_perm.[c]uid** and the process's effective group ID does not match **msg\_perm.[c]gid** and the appropriate bit of the "other" portion (06) of **msg\_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

### Semaphore Identifier

A semaphore identifier (*semid*) is a unique positive integer created by a *semget*(2) system call. Each *semid* has a set of semaphores and a data structure associated with it. The data structure is referred to as *semid\_ds* and contains the following members:

```
struct ipc_perm sem_perm; /* operation permission struct */
ushort sem_nsems; /* number of sems in set */
time_t sem_otime; /* last operation time */
time_t sem_ctime; /* last change time */
/* Times measured in secs since */
/* 00:00:00 GMT, Jan. 1, 1970 */
```

**Sem\_perm** is a *ipc\_perm* structure that specifies the semaphore operation permission (see below). This structure includes the following members:

```
ushort cuid; /* creator user id */
ushort cgid; /* creator group id */
ushort uid; /* user id */
ushort gid; /* group id */
ushort mode; /* r/a permission */
```

The value of **sem\_nsems** is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as

a *sem\_num*. *Sem\_num* values run sequentially from 0 to the value of *sem\_nsems* minus 1. *Sem\_otime* is the time of the last *semop(2)* operation, and *sem\_ctime* is the time of the last *semctl(2)* operation that changed a member of the above structure.

A semaphore is a data structure that contains the following members:

```

ushort  semval;      /* semaphore value */
short   sempid;     /* pid of last operation */
ushort  semncnt;    /* # awaiting semval > cval */
ushort  semzcnt;    /* # awaiting semval = 0 */

```

*Semval* is a non-negative integer. *Sempid* is equal to the process ID of the last process that performed a semaphore operation on this semaphore. *Semncnt* is a count of the number of processes that are currently suspended awaiting this semaphore's *semval* to become greater than its current value. *Semzcnt* is a count of the number of processes that are currently suspended awaiting this semaphore's *semval* to become zero.

### Semaphore Operation Permissions.

In the *semop(2)* and *semctl(2)* system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

|       |                       |
|-------|-----------------------|
| 00400 | Read by user          |
| 00200 | Alter by user         |
| 00060 | Read, Alter by group  |
| 00006 | Read, Alter by others |

Read and Alter permissions on a *semid* are granted to a process if one or more of the following are true:

The process's effective user ID is super-user.

The process's effective user ID matches *sem\_perm.[c]uid* in the data structure associated with *semid* and the appropriate bit of the "user" portion (0600) of *sem\_perm.mode* is set.

The process's effective user ID does not match *sem\_perm.[c]uid* and the process's effective group ID matches *sem\_perm.[c]gid* and the appropriate bit of the "group" portion (060) of *sem\_perm.mode* is set.

The process's effective user ID does not match *sem\_perm.[c]uid* and the process's effective group ID does not match *sem\_perm.[c]gid* and the appropriate bit of the "other" portion (06) of *sem\_perm.mode* is set.

Otherwise, the corresponding permissions are denied.

### Shared Memory Identifier

A shared memory identifier (*shmid*) is a unique positive integer created by a *shmget(2)* system call. Each *shmid* has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. The data structure is referred to as *shmid\_ds* and contains the following members:

```

struct  ipc_perm shm_perm; /* operation permission struct */
int     shm_segsz; /* size of segment */
ushort  shm_cpid; /* creator pid */
ushort  shm_lpid; /* pid of last operation */
short   shm_nattch; /* number of current attaches */
time_t  shm_atime; /* last attach time */
time_t  shm_dtime; /* last detach time */

```

```

time_t  shm_ctime;          /* last change time */
                               /* Times measured in secs since */
                               /* 00:00:00 GMT, Jan. 1, 1970 */

```

**Shm\_perm** is a `ipc_perm` structure that specifies the shared memory operation permission (see below). This structure includes the following members:

```

ushort  cuid;              /* creator user id */
ushort  cgid;              /* creator group id */
ushort  uid;               /* user id */
ushort  gid;               /* group id */
ushort  mode;              /* r/w permission */

```

**Shm\_segsz** specifies the size of the shared memory segment. **Shm\_cpuid** is the process id of the process that created the shared memory identifier. **Shm\_lpid** is the process id of the last process that performed a `shmop(2)` operation. **Shm\_nattch** is the number of processes that currently have this segment attached. **Shm\_atime** is the time of the last `shmat` operation, **shm\_dtime** is the time of the last `shmdt` operation, and **shm\_ctime** is the time of the last `shmctl(2)` operation that changed one of the members of the above structure.

### Shared Memory Operation Permissions.

In the `shmop(2)` and `shmctl(2)` system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

|       |                       |
|-------|-----------------------|
| 00400 | Read by user          |
| 00200 | Write by user         |
| 00060 | Read, Write by group  |
| 00006 | Read, Write by others |

Read and Write permissions on a `shmid` are granted to a process if one or more of the following are true:

The process's effective user ID is super-user.

The process's effective user ID matches `shm_perm.[c]uid` in the data structure associated with `shmid` and the appropriate bit of the "user" portion (0600) of `shm_perm.mode` is set.

The process's effective user ID does not match `shm_perm.[c]uid` and the process's effective group ID matches `shm_perm.[c]gid` and the appropriate bit of the "group" portion (060) of `shm_perm.mode` is set.

The process's effective user ID does not match `shm_perm.[c]uid` and the process's effective group ID does not match `shm_perm.[c]gid` and the appropriate bit of the "other" portion (06) of `shm_perm.mode` is set.

Otherwise, the corresponding permissions are denied.

### SEE ALSO

`intro(3)`.

**NAME**

access — determine accessibility of a file

**SYNOPSIS**

```
int access (path, amode)
char *path;
int amode;
```

**DESCRIPTION**

*Path* points to a path name naming a file. *Access* checks the named file for accessibility according to the bit pattern contained in *amode*, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID. The bit pattern contained in *amode* is constructed as follows:

```
04    read
02    write
01    execute (search)
00    check existence of file
```

Access to the file is denied if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

Read, write, or execute (search) permission is requested for a null path name. [ENOENT]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

Write access is requested for a file on a read-only file system. [EROFS]

Write access is requested for a pure procedure (shared text) file that is being executed. [ETXTBSY]

Permission bits of the file mode do not permit the requested access. [EACCES]

*Path* points outside the process's allocated address space. [EFAULT]

The owner of a file has permission checked with respect to the "owner" read, write, and execute mode bits, members of the file's group other than the owner have permissions checked with respect to the "group" mode bits, and all others have permissions checked with respect to the "other" mode bits.

**RETURN VALUE**

If the requested access is permitted, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

chmod(2), stat(2).



**NAME**

**acct** — enable or disable process accounting

**SYNOPSIS**

```
int acct (path)
char *path;
```

**DESCRIPTION**

*Acct* is used to enable or disable the system's process accounting routine. If the routine is enabled, an accounting record will be written on an accounting file for each process that terminates. Termination can be caused by one of two things: an *exit* call or a signal; see *exit(2)* and *signal(2)*. The effective user ID of the calling process must be super-user to use this call.

*Path* points to a path name naming the accounting file. The accounting file format is given in *acct(4)*.

The accounting routine is enabled if *path* is non-zero and no errors occur during the system call. It is disabled if *path* is zero and no errors occur during the system call.

*Acct* will fail if one or more of the following are true:

The effective user ID of the calling process is not super-user. [EPERM]

An attempt is being made to enable accounting when it is already enabled. [EBUSY]

A component of the path prefix is not a directory. [ENOTDIR]

One or more components of the accounting file's path name do not exist. [ENOENT]

A component of the path prefix denies search permission. [EACCES]

The file named by *path* is not an ordinary file. [EACCES]

*Mode* permission is denied for the named accounting file. [EACCES]

The named file is a directory. [EISDIR]

The named file resides on a read-only file system. [EROFS]

*Path* points to an illegal address. [EFAULT]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

*acct(4)*.

**NAME**

alarm — set a process's alarm clock

**SYNOPSIS**

**unsigned alarm (sec)**  
**unsigned sec;**

**DESCRIPTION**

*Alarm* instructs the calling process's alarm clock to send the signal SIGALRM to the calling process after the number of real time seconds specified by *sec* have elapsed; see *signal(2)*.

Alarm requests are not stacked; successive calls reset the calling process's alarm clock.

If *sec* is 0, any previously made alarm request is canceled.

**RETURN VALUE**

*Alarm* returns the amount of time previously remaining in the calling process's alarm clock.

**SEE ALSO**

pause(2), signal(2).

## NAME

*brk*, *sbrk* — change data segment space allocation

## SYNOPSIS

```
int brk (endds)
char *endds;

char *sbrk (incr)
int incr;
```

## DESCRIPTION

*Brk* and *sbrk* are used to change dynamically the amount of space allocated for the calling process's data segment; see *exec(2)*. The change is made by resetting the process's break value and allocating the appropriate amount of space. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases. The newly allocated space is set to zero.

*Brk* sets the break value to *endds* and changes the allocated space accordingly.

*Sbrk* adds *incr* bytes to the break value and changes the allocated space accordingly. *Incr* can be negative, in which case the amount of allocated space is decreased.

*Brk* and *sbrk* will fail without making any change in the allocated space if one or more of the following are true:

Such a change would result in more space being allocated than is allowed by a system-imposed maximum (see *ulimit(2)*). [ENOMEM]

Such a change would result in the break value being greater than or equal to the start address of any attached shared memory segment (see *shmop(2)*).

## RETURN VALUE

Upon successful completion, *brk* returns a value of 0 and *sbrk* returns the old break value. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## SEE ALSO

*exec(2)*.

**NAME**

`chdir` — change working directory

**SYNOPSIS**

```
int chdir (path)
char *path;
```

**DESCRIPTION**

*Path* points to the path name of a directory. *Chdir* causes the named directory to become the current working directory, the starting point for path searches for path names not beginning with `/`.

*Chdir* will fail and the current working directory will be unchanged if one or more of the following are true:

A component of the path name is not a directory. [ENOTDIR]

The named directory does not exist. [ENOENT]

Search permission is denied for any component of the path name. [EACCES]

*Path* points outside the process's allocated address space. [EFAULT]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of `-1` is returned and *errno* is set to indicate the error.

**SEE ALSO**

`chroot(2)`.

## NAME

chmod — change mode of file

## SYNOPSIS

```
int chmod (path, mode)
char *path;
int mode;
```

## DESCRIPTION

*Path* points to a path name naming a file. *Chmod* sets the access permission portion of the named file's mode according to the bit pattern contained in *mode*.

Access permission bits are interpreted as follows:

```
04000 Set user ID on execution.
02000 Set group ID on execution.
01000 Save text image after execution
00400 Read by owner
00200 Write by owner
00100 Execute (or search if a directory) by owner
00070 Read, write, execute (search) by group
00007 Read, write, execute (search) by others
```

The effective user ID of the process must match the owner of the file or be super-user to change the mode of a file.

If the effective user ID of the process is not super-user, mode bit 01000 (save text image on execution) is cleared.

If the effective user ID of the process is not super-user or the effective group ID of the process does not match the group ID of the file, mode bit 02000 (set group ID on execution) is cleared.

If an executable file is prepared for sharing then mode bit 01000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus, when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time.

*Chmod* will fail and the file mode will be unchanged if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The effective user ID does not match the owner of the file and the effective user ID is not super-user. [EPERM]

The named file resides on a read-only file system. [EROFS]

*Path* points outside the process's allocated address space. [EFAULT]

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## SEE ALSO

chown(2), mknod(2).

**NAME**

**chown** — change owner and group of a file

**SYNOPSIS**

```
int chown (path, owner, group)
char *path;
int owner, group;
```

**DESCRIPTION**

*Path* points to a path name naming a file. The owner ID and group ID of the named file are set to the numeric values contained in *owner* and *group* respectively.

Only processes with effective user ID equal to the file owner or super-user may change the ownership of a file.

If *chown* is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

*Chown* will fail and the owner and group of the named file will remain unchanged if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The effective user ID does not match the owner of the file and the effective user ID is not super-user. [EPERM]

The named file resides on a read-only file system. [EROFS]

*Path* points outside the process's allocated address space. [EFAULT]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

**chmod(2).**

**NAME**

chroot — change root directory

**SYNOPSIS**

```
int chroot (path)
char *path;
```

**DESCRIPTION**

*Path* points to a path name naming a directory. *Chroot* causes the named directory to become the root directory, the starting point for path searches for path names beginning with /.

The effective user ID of the process must be super-user to change the root directory.

The .. entry in the root directory is interpreted to mean the root directory itself. Thus, .. can not be used to access files outside the subtree rooted at the root directory.

*Chroot* will fail and the root directory will remain unchanged if one or more of the following are true:

Any component of the path name is not a directory. [ENOTDIR]

The named directory does not exist. [ENOENT]

The effective user ID is not super-user. [EPERM]

*Path* points outside the process's allocated address space. [EFAULT]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

chdir(2).

**NAME**

close — close a file descriptor

**SYNOPSIS**

```
int close (fildes)
int fildes;
```

**DESCRIPTION**

*Fildes* is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. *Close* closes the file descriptor indicated by *fildes*.

*Close* will fail if *fildes* is not a valid open file descriptor. [EBADF]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

*creat*(2), *dup*(2), *exec*(2), *fcntl*(2), *open*(2), *pipe*(2).



**NAME**

*creat* — create a new file or rewrite an existing one

**SYNOPSIS**

```
int creat (path, mode)
char *path;
int mode;
```

**DESCRIPTION**

*Creat* creates a new ordinary file or prepares to rewrite an existing file named by the path name pointed to by *path*.

If the file exists, the length is truncated to 0 and the mode and owner are unchanged. Otherwise, the file's owner ID is set to the process's effective user ID, the file's group ID is set to the process's effective group ID, and the low-order 12 bits of the file mode are set to the value of *mode* modified as follows:

All bits set in the process's file mode creation mask are cleared. See *umask*(2).

The "save text image after execution bit" of the mode is cleared. See *chmod*(2).

Upon successful completion, a non-negative integer, namely the file descriptor, is returned and the file is open for writing, even if the mode does not permit writing. The file pointer is set to the beginning of the file. The file descriptor is set to remain open across *exec* system calls. See *fcntl*(2). No process may have more than 20 files open simultaneously. A new file may be created with a mode that forbids writing.

*Creat* will fail if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

A component of the path prefix does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The path name is null. [ENOENT]

The file does not exist and the directory in which the file is to be created does not permit writing. [EACCES]

The named file resides or would reside on a read-only file system. [EROFS]

The file is a pure procedure (shared text) file that is being executed. [ETXTBSY]

The file exists and write permission is denied. [EACCES]

The named file is an existing directory. [EISDIR]

Twenty (20) file descriptors are currently open. [EMFILE]

*Path* points outside the process's allocated address space. [EFAULT]

**RETURN VALUE**

Upon successful completion, a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

*close*(2), *dup*(2), *lseek*(2), *open*(2), *read*(2), *umask*(2), *write*(2).



**NAME**

`dup` — duplicate an open file descriptor

**SYNOPSIS**

```
int dup (fildes)
int fildes;
```

**DESCRIPTION**

*Fildes* is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. *Dup* returns a new file descriptor having the following in common with the original:

Same open file (or pipe).

Same file pointer. (i.e., both file descriptors share one file pointer.)

Same access mode (read, write or read/write).

The new file descriptor is set to remain open across *exec* system calls. See *fcntl(2)*.

The file descriptor returned is the lowest one available.

*Dup* will fail if one or more of the following are true:

*Fildes* is not a valid open file descriptor. [EBADF]

Twenty (20) file descriptors are currently open. [EMFILE]

**RETURN VALUE**

Upon successful completion a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

**SEE ALSO**

*creat(2)*, *close(2)*, *exec(2)*, *fcntl(2)*, *open(2)*, *pipe(2)*.

## NAME

execl, execv, execl, execve, execlp, execvp — execute a file

## SYNOPSIS

```
int execl (path, arg0, arg1, ..., argn, 0)
char *path, *arg0, *arg1, ..., *argn;

int execv (path, argv)
char *path, *argv[ ];

int execl (path, arg0, arg1, ..., argn, 0, envp)
char *path, *arg0, *arg1, ..., *argn, *envp[ ];

int execve (path, argv, envp)
char *path, *argv[ ], *envp[ ];

int execlp (file, arg0, arg1, ..., argn, 0)
char *file, *arg0, *arg1, ..., *argn;

int execvp (file, argv)
char *file, *argv[ ];
```

## DESCRIPTION

*Exec* in all its forms transforms the calling process into a new process. The new process is constructed from an ordinary, executable file called the *new process file*. This file consists of a header (see *a.out(4)*), a text segment, and a data segment. The data segment contains an initialized portion and an uninitialized portion (bss). There can be no return from a successful *exec* because the calling process is overlaid by the new process.

When a C program is executed, it is called as follows:

```
main (argc, argv, envp)
int argc;
char **argv, **envp;
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. As indicated, *argc* is conventionally at least one and the first member of the array points to a string containing the name of the file.

*Path* points to a path name that identifies the new process file.

*File* points to the new process file. The path prefix for this file is obtained by a search of the directories passed as the *environment* line "PATH =" (see *environ(5)*). The environment is supplied by the shell (see *sh(1)*).

*Arg0*, *arg1*, ..., *argn* are pointers to null-terminated character strings. These strings constitute the argument list available to the new process. By convention, at least *arg0* must be present and point to a string that is the same as *path* (or its last component).

*Argv* is an array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process. By convention, *argv* must have at least one member, and it must point to a string that is the same as *path* (or its last component). *Argv* is terminated by a null pointer.

*Envp* is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process. *Envp* is terminated by a null pointer. For *execl* and *execv*, the C run-time start-off routine places a pointer to the calling process's environment in the global cell:

```
extern char **environ;
```

and it is used to pass the calling process's environment to the new process.

File descriptors open in the calling process remain open in the new process, except for those whose close-on-exec flag is set; see *fcntl(2)*. For those file descriptors that remain open, the file pointer is unchanged.

Signals set to terminate the calling process will be set to terminate the new process. Signals set to be ignored by the calling process will be set to be ignored by the new process. Signals set to be caught by the calling process will be set to terminate new process; see *signal(2)*.

If the set-user-ID mode bit of the new process file is set (see *chmod(2)*), *exec* sets the effective user ID of the new process to the owner ID of the new process file. Similarly, if the set-group-ID mode bit of the new process file is set, the effective group ID of the new process is set to the group ID of the new process file. The real user ID and real group ID of the new process remain the same as those of the calling process.

The shared memory segments attached to the calling process will not be attached to the new process (see *shmop(2)*).

Profiling is disabled for the new process; see *profil(2)*.

The new process also inherits the following attributes from the calling process:

- nice value (see *nice(2)*)
- process ID
- parent process ID
- process group ID
- semadj values (see *semop(2)*)
- tty group ID (see *exit(2)* and *signal(2)*)
- trace flag (see *ptrace(2)* request 0)
- time left until an alarm clock signal (see *alarm(2)*)
- current working directory
- root directory
- file mode creation mask (see *umask(2)*)
- file size limit (see *ulimit(2)*)
- utime*, *stime*, *cutime*, and *cstime* (see *times(2)*)

*Exec* will fail and return to the calling process if one or more of the following are true:

One or more components of the new process file's path name do not exist. [ENOENT]

A component of the new process file's path prefix is not a directory. [ENOTDIR]

Search permission is denied for a directory listed in the new process file's path prefix. [EACCES]

The new process file is not an ordinary file. [EACCES]

The new process file mode denies execution permission. [EACCES]

The *exec* is not an *execlp* or *execvp*, and the new process file has the appropriate access permission but an invalid magic number in its header. [ENOEXEC]

The new process file is a pure procedure (shared text) file that is currently open for writing by some process. [ETXTBSY]

The new process requires more memory than is allowed by the system-imposed maximum MAXMEM. [ENOMEM]

The number of bytes in the new process's argument list is greater than the system-imposed limit of 5120 bytes. [E2BIG]

The new process file is not as long as indicated by the size values in its header. [EFAULT]

*Path*, *argv*, or *envp* point to an illegal address. [EFAULT]

**RETURN VALUE**

If *exec* returns to the calling process an error has occurred; the return value will be  $-1$  and *errno* will be set to indicate the error.

**SEE ALSO**

*exit*(2), *fork*(2), *environ*(5).

**NAME**

`exit`, `_exit` — terminate process

**SYNOPSIS**

```
void exit (status)
int status;
void _exit (status)
int status;
```

**DESCRIPTION**

*Exit* terminates the calling process with the following consequences:

All of the file descriptors open in the calling process are closed.

If the parent process of the calling process is executing a *wait*, it is notified of the calling process's termination and the low order eight bits (i.e., bits 0377) of *status* are made available to it; see *wait(2)*.

If the parent process of the calling process is not executing a *wait*, the calling process is transformed into a zombie process. A *zombie process* is a process that only occupies a slot in the process table, it has no other space allocated either in user or kernel space. The process table slot that it occupies is partially overlaid with time accounting information (see `<sys/proc.h>`) to be used by *times*.

The parent process ID of all of the calling process's existing child processes and zombie processes is set to 1. This means the initialization process (see *intro(2)*) inherits each of these processes.

Each attached shared memory segment is detached and the value of `shm_nattach` in the data structure associated with its shared memory identifier is decremented by 1.

For each semaphore for which the calling process has set a `semadj` value (see *semop(2)*), that `semadj` value is added to the `semval` of the specified semaphore.

If the process has a process, text, or data lock, an *unlock* is performed (see *plock(2)*).

An accounting record is written on the accounting file if the system's accounting routine is enabled; see *acct(2)*.

If the process ID, tty group ID, and process group ID of the calling process are equal, the `SIGHUP` signal is sent to each processes that has a process group ID equal to that of the calling process.

The C function *exit* may cause cleanup actions before the process exits. The function *\_exit* circumvents all cleanup.

**SEE ALSO**

*signal(2)*, *wait(2)*.

**WARNING**

See *WARNING* in *signal(2)*.

## NAME

*fcntl* — file control

## SYNOPSIS

```
#include <fcntl.h>
int fcntl (fildes, cmd, arg)
int fildes, cmd, arg;
```

## DESCRIPTION

*Fcntl* provides for control over open files. *Fildes* is an open file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call.

The *cmds* available are:

- F\_DUPFD** Return a new file descriptor as follows:
- Lowest numbered available file descriptor greater than or equal to *arg*.
  - Same open file (or pipe) as the original file.
  - Same file pointer as the original file (i.e., both file descriptors share one file pointer).
  - Same access mode (read, write or read/write).
  - Same file status flags (i.e., both file descriptors share the same file status flags).
  - The close-on-exec flag associated with the new file descriptor is set to remain open across *exec(2)* system calls.
- F\_GETFD** Get the close-on-exec flag associated with the file descriptor *fildes*. If the low-order bit is 0 the file will remain open across *exec*, otherwise the file will be closed upon execution of *exec*.
- F\_SETFD** Set the close-on-exec flag associated with *fildes* to the low-order bit of *arg* (0 or 1 as above).
- F\_GETFL** Get *file* status flags.
- F\_SETFL** Set *file* status flags to *arg*. Only certain flags can be set; see *fcntl(5)*.

*Fcntl* will fail if one or more of the following are true:

*Fildes* is not a valid open file descriptor. [EBADF]

*Cmd* is **F\_DUPFD** and 20 file descriptors are currently open. [EMFILE]

*Cmd* is **F\_DUPFD** and *arg* is negative or greater than 20. [EINVAL]

## RETURN VALUE

Upon successful completion, the value returned depends on *cmd* as follows:

- F\_DUPFD** A new file descriptor.
- F\_GETFD** Value of flag (only the low-order bit is defined).
- F\_SETFD** Value other than -1.
- F\_GETFL** Value of file flags.
- F\_SETFL** Value other than -1.

Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## SEE ALSO

*close(2)*, *exec(2)*, *open(2)*, *fcntl(5)*.

## NAME

fork — create a new process

## SYNOPSIS

```
int fork ( )
```

## DESCRIPTION

*Fork* causes creation of a new process. The new process (child process) is an exact copy of the calling process (parent process). This means the child process inherits the following attributes from the parent process:

- environment
- close-on-exec flag (see *exec(2)*)
- signal handling settings (i.e., *SIG\_DFL*, *SIG\_IGN*, function address)
- set-user-ID mode bit
- set-group-ID mode bit
- profiling on/off status
- nice value (see *nice(2)*)
- all attached shared memory segments (see *shmop(2)*)
- process group ID
- tty group ID (see *exit(2)* and *signal(2)*)
- trace flag (see *ptrace(2)* request 0)
- time left until an alarm clock signal (see *alarm(2)*)
- current working directory
- root directory
- file mode creation mask (see *umask(2)*)
- file size limit (see *ulimit(2)*)

The child process differs from the parent process in the following ways:

The child process has a unique process ID.

The child process has a different parent process ID (i.e., the process ID of the parent process).

The child process has its own copy of the parent's file descriptors. Each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent.

All *semadj* values are cleared (see *semop(2)*).

Process locks, text locks and data locks are not inherited by the child (see *plock(2)*).

The child process's *utime*, *stime*, *cutime*, and *cstime* are set to 0.

*Fork* will fail and no child process will be created if one or more of the following are true:

The system-imposed limit on the total number of processes under execution would be exceeded. [EAGAIN]

The system-imposed limit on the total number of processes under execution by a single user would be exceeded. [EAGAIN]

## RETURN VALUE

Upon successful completion, *fork* returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of -1 is returned to the parent process, no child process is created, and *errno* is set to indicate the error.

## SEE ALSO

*exec(2)*, *times(2)*, *wait(2)*.



**NAME**

`getpid`, `getpgrp`, `getppid` — get process, process group, and parent process IDs

**SYNOPSIS**

**int** `getpid` ( )

**int** `getpgrp` ( )

**int** `getppid` ( )

**DESCRIPTION**

*Getpid* returns the process ID of the calling process.

*Getpgrp* returns the process group ID of the calling process.

*Getppid* returns the parent process ID of the calling process.

**SEE ALSO**

`exec(2)`, `fork(2)`, `intro(2)`, `setpgrp(2)`, `signal(2)`.

**NAME**

**getuid, geteuid, getgid, getegid** — get real user, effective user, real group, and effective group IDs

**SYNOPSIS**

**int getuid ( )**

**int geteuid ( )**

**int getgid ( )**

**int getegid ( )**

**DESCRIPTION**

*Getuid* returns the real user ID of the calling process.

*Geteuid* returns the effective user ID of the calling process.

*Getgid* returns the real group ID of the calling process.

*Getegid* returns the effective group ID of the calling process.

**SEE ALSO**

intro(2), setuid(2).

**NAME**

*ioctl* — control device

**SYNOPSIS**

***ioctl* (*fildes*, *request*, *arg*)**

**DESCRIPTION**

*Ioctl* performs a variety of functions on character special files (devices). The writeups of various devices in Section 7 discuss how *ioctl* applies to them.

*Ioctl* will fail if one or more of the following are true:

*Fildes* is not a valid open file descriptor. [EBADF]

*Fildes* is not associated with a character special device. [ENOTTY]

*Request* or *arg* is not valid. See Section 7. [EINVAL]

**RETURN VALUE**

If an error has occurred, a value of  $-1$  is returned and *errno* is set to indicate the error.

**SEE ALSO**

*termio(7)* in the *UNIX System Administrator's Manual*.

**NAME**

**kill** — send a signal to a process or a group of processes

**SYNOPSIS**

```
int kill (pid, sig)
int pid, sig;
```

**DESCRIPTION**

*Kill* sends a signal to a process or a group of processes. The process or group of processes to which the signal is to be sent is specified by *pid*. The signal that is to be sent is specified by *sig* and is either one from the list given in *signal(2)*, or 0. If *sig* is 0 (the null signal), error checking is performed but no signal is actually sent. This can be used to check the validity of *pid*.

The real or effective user ID of the sending process must match the real or effective user ID of the receiving process unless, the effective user ID of the sending process is super-user.

The processes with a process ID of 0 and a process ID of 1 are special processes (see *intro(2)*) and will be referred to below as *proc0* and *proc1* respectively.

If *pid* is greater than zero, *sig* will be sent to the process whose process ID is equal to *pid*. *Pid* may equal 1.

If *pid* is 0, *sig* will be sent to all processes excluding *proc0* and *proc1* whose process group ID is equal to the process group ID of the sender.

If *pid* is -1 and the effective user ID of the sender is not super-user, *sig* will be sent to all processes excluding *proc0* and *proc1* whose real user ID is equal to the effective user ID of the sender.

If *pid* is -1 and the effective user ID of the sender is super-user, *sig* will be sent to all processes excluding *proc0* and *proc1*.

If *pid* is negative but not -1, *sig* will be sent to all processes whose process group ID is equal to the absolute value of *pid*.

*Kill* will fail and no signal will be sent if one or more of the following are true:

*Sig* is not a valid signal number. [EINVAL]

No process can be found corresponding to that specified by *pid*. [ESRCH]

The user ID of the sending process is not super-user, and its real or effective user ID does not match the real or effective user ID of the receiving process. [EPERM]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

*kill(1)*, *getpid(2)*, *setpgrp(2)*, *signal(2)*.

**NAME**

link — link to a file

**SYNOPSIS**

```
int link (path1, path2)
char *path1, *path2;
```

**DESCRIPTION**

*Path1* points to a path name naming an existing file. *Path2* points to a path name naming the new directory entry to be created. *Link* creates a new link (directory entry) for the existing file.

*Link* will fail and no link will be created if one or more of the following are true:

A component of either path prefix is not a directory. [ENOTDIR]

A component of either path prefix does not exist. [ENOENT]

A component of either path prefix denies search permission. [EACCES]

The file named by *path1* does not exist. [ENOENT]

The link named by *path2* exists. [EEXIST]

The file named by *path1* is a directory and the effective user ID is not super-user. [EPERM]

The link named by *path2* and the file named by *path1* are on different logical devices (file systems). [EXDEV]

*Path2* points to a null path name. [ENOENT]

The requested link requires writing in a directory with a mode that denies write permission. [EACCES]

The requested link requires writing in a directory on a read-only file system. [EROFS]

*Path* points outside the process's allocated address space. [EFAULT]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

unlink(2).

**NAME**

`lseek` — move read/write file pointer

**SYNOPSIS**

```
long lseek (fildes, offset, whence)
int fildes;
long offset;
int whence;
```

**DESCRIPTION**

*Fildes* is a file descriptor returned from a *creat*, *open*, *dup*, or *fcntl* system call. *Lseek* sets the file pointer associated with *fildes* as follows:

If *whence* is 0, the pointer is set to *offset* bytes.

If *whence* is 1, the pointer is set to its current location plus *offset*.

If *whence* is 2, the pointer is set to the size of the file plus *offset*.

Upon successful completion, the resulting pointer location as measured in bytes from the beginning of the file is returned.

*Lseek* will fail and the file pointer will remain unchanged if one or more of the following are true:

*Fildes* is not an open file descriptor. [EBADF]

*Fildes* is associated with a pipe or fifo. [ESPIPE]

*Whence* is not 0, 1 or 2. [EINVAL and SIGSYS signal]

The resulting file pointer would be negative. [EINVAL]

Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.

**RETURN VALUE**

Upon successful completion, a non-negative integer indicating the file pointer value is returned. Otherwise, a value of `-1` is returned and *errno* is set to indicate the error.

**SEE ALSO**

`creat(2)`, `dup(2)`, `fcntl(2)`, `open(2)`.

**NAME**

maus — multiple-access-user-space (shared memory) operations

**SYNOPSIS**

```
#include <sys/fcntl.h>

int getmaus (path, oflag)
char *path;
int oflag;

int freemaus (mausdes)
int mausdes;

char *enabmaus (mausdes)
int mausdes;

int dismaus (saddr)
char *saddr;

char *switmaus (mausdes, saddr)
int mausdes;
char *saddr;
```

**DESCRIPTION**

MAUS (Multiple Access User Space) is a dedicated portion of physical memory that is subdivided into logical subsections. These subsections can be attached to the calling process's data segment or released from its data segment with the following calls.

*Path* points to a path name naming a special file that is one of the MAUS logical subsections. *Getmaus* opens a maus descriptor for the named file and sets the file status flag according to the value of *oflag*. *Oflag* is one of the following:

- O\_RDONLY** Open for reading only.
- O\_WRONLY** Open for writing only.
- O\_RDWR** Open for reading and writing.

No process may have more than eight (8) maus descriptors open simultaneously.

The named file is opened unless one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

The named file is not a maus special file. [EINVAL]

A component of the path prefix denies search permission. [EACCES]

*Oflag* permission is denied for the named file. [EACCES]

Eight (8) maus descriptors are currently open. [EMFILE]

The MAUS area associated with the special file does not exist. [ENXIO]

*Path* points to an illegal address. [EFAULT]

*Freemaus* closes the maus descriptor specified by *mausdes*. Note that if a maus descriptor has been enabled (see *enabmaus* below) it may still be closed: a MAUS file remains attached to a process's data segment until a *dismaus* (see below) is used to free it.

*Freemaus* will fail if *mausdes* is not a valid open maus descriptor. [EBADF]

*Enabmaus* attaches the MAUS file associated with *mausdes* to the data segment of the calling process. The file is attached starting at the first available 8k-byte boundary address beyond the current break value (see *brk(2)*). Note that multiple *enabmaus* calls can be made with the same *maus* descriptor. Each call will attach the file at a different 8k-byte boundary address.

*Enabmaus* will fail and not attach the MAUS file if one or more of the following are true:

*Mausdes* is not a valid open *maus* descriptor. [EBADF]

No more 8k-byte boundary starting addresses are available. [ENOMEM]

*Dismaus* frees from the calling process's data segment the MAUS file that starts at the data segment address given by (*saddr* - (*saddr* modulus 8192)).

*Dismaus* will fail and not free the MAUS file if (*saddr* - (*saddr* modulus 8192)) is not the data segment starting address of a MAUS file. [EINVAL]

*Switmaus* attaches the MAUS file associated with *mausdes* to the data segment of the calling process. The file is attached starting at the address given by (*saddr* - (*saddr* modulus 8192)).

*Switmaus* will fail if one or more of the following are true:

*Mausdes* is not a valid open *maus* descriptor. [EBADF]

The value of (*saddr* - (*saddr* modulus 8192)) is not a legal 8k-byte boundary address above the current break value. [EINVAL]

## RETURN VALUES

Upon successful completion, the return value is as follows:

*Getmaus* returns a non-negative integer, namely a *maus* descriptor.

*Freemaus* returns a value of 0.

*Enabmaus* returns the data segment starting address of the attached MAUS file.

*Dismaus* and *switmaus* return the *maus* descriptor previously associated with the data segment starting address given by (*saddr* - (*saddr* modulus 8192)) if one exists. Otherwise, a value of -2 is returned.

On other than successful completion, a value of -1 is returned with *errno* set to indicate the error.



**NAME**

**mknod** — make a directory, or a special or ordinary file

**SYNOPSIS**

```
int mknod (path, mode, dev)
char *path;
int mode, dev;
```

**DESCRIPTION**

*Mknod* creates a new file named by the path name pointed to by *path*. The mode of the new file is initialized from *mode*. Where the value of *mode* is interpreted as follows:

```
0170000 file type; one of the following:
    0010000 fifo special
    0020000 character special
    0040000 directory
    0060000 block special
    0100000 or 0000000 ordinary file
0004000 set user ID on execution
0002000 set group ID on execution
0001000 save text image after execution
0000777 access permissions; constructed from the following
    0000400 read by owner
    0000200 write by owner
    0000100 execute (search on directory) by owner
    0000070 read, write, execute (search) by group
    0000007 read, write, execute (search) by others
```

The file's owner ID is set to the process's effective user ID. The file's group ID is set to the process's effective group ID.

Values of *mode* other than those above are undefined and should not be used. The low-order 9 bits of *mode* are modified by the process's file mode creation mask; all bits set in the process's file mode creation mask are cleared. See *umask(2)*. If *mode* indicates a block or character special file, *dev* is a configuration dependent specification of a character or block I/O device. If *mode* does not indicate a block special or character special device, *dev* is ignored.

*Mknod* may be invoked only by the super-user for file types other than FIFO special.

*Mknod* will fail and the new file will not be created if one or more of the following are true:

- The process's effective user ID is not super-user. [EPERM]
- A component of the path prefix is not a directory. [ENOTDIR]
- A component of the path prefix does not exist. [ENOENT]
- The directory in which the file is to be created is located on a read-only file system. [EROFS]
- The named file exists. [EEXIST]
- Path* points outside the process's allocated address space. [EFAULT]

**RETURN VALUE**

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

*mkdir(1)*, *chmod(2)*, *exec(2)*, *umask(2)*, *fs(4)*.

**NAME**

mount — mount a file system

**SYNOPSIS**

```
int mount (spec, dir, rflag)
char *spec, *dir;
int rflag;
```

**DESCRIPTION**

*Mount* requests that a removable file system contained on the block special file identified by *spec* be mounted on the directory identified by *dir*. *Spec* and *dir* are pointers to path names.

Upon successful completion, references to the file *dir* will refer to the root directory on the mounted file system.

The low-order bit of *rflag* is used to control write permission on the mounted file system; if 1, writing is forbidden, otherwise writing is permitted according to individual file accessibility.

*Mount* may be invoked only by the super-user.

*Mount* will fail if one or more of the following are true:

The effective user ID is not super-user. [EPERM]

Any of the named files does not exist. [ENOENT]

A component of a path prefix is not a directory. [ENOTDIR]

*Spec* is not a block special device. [ENOTBLK]

The device associated with *spec* does not exist. [ENXIO]

*Dir* is not a directory. [ENOTDIR]

*Spec* or *dir* points outside the process's allocated address space. [EFAULT]

*Dir* is currently mounted on, is someone's current working directory or is otherwise busy. [EBUSY]

The device associated with *spec* is currently mounted. [EBUSY]

**RETURN VALUE**

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

umount(2).

## NAME

`msgctl` — message control operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl (msqid, cmd, buf)
int msqid, cmd;
struct msqid_ds *buf;
```

## DESCRIPTION

*Msgctl* provides a variety of message control operations as specified by *cmd*. The following *cmds* are available:

- IPC\_STAT** Place the current value of each member of the data structure associated with *msqid* into the structure pointed to by *buf*. The contents of this structure are defined in *intro(2)*. {READ}
- IPC\_SET** Set the value of the following members of the data structure associated with *msqid* to the corresponding value found in the structure pointed to by *buf*:
- `msg_perm.uid`
  - `msg_perm.gid`
  - `msg_perm.mode` /\* only low 9 bits \*/
  - `msg_qbytes`

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super user or to the value of `msg_perm.uid` in the data structure associated with *msqid*. Only super user can raise the value of `msg_qbytes`.

- IPC\_RMID** Remove the message queue identifier specified by *msqid* from the system and destroy the message queue and data structure associated with it. This *cmd* can only be executed by a process that has an effective user ID equal to either that of super user or to the value of `msg_perm.uid` in the data structure associated with *msqid*.

*Msgctl* will fail if one or more of the following are true:

*Msqid* is not a valid message queue identifier. [EINVAL]

*Cmd* is not a valid command. [EINVAL]

*Cmd* is equal to `IPC_STAT` and {READ} operation permission is denied to the calling process (see *intro(2)*). [EACCESS]

*Cmd* is equal to `IPC_RMID` or `IPC_SET` and the effective user ID of the calling process is not equal to that of super user and it is not equal to the value of `msg_perm.uid` in the data structure associated with *msqid*. [EPERM]

*Cmd* is equal to `IPC_SET`, an attempt is being made to increase to the value of `msg_qbytes`, and the effective user ID of the calling process is not equal to that of super user. [EPERM]

*Buf* points to an illegal address. [EFAULT]

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## SEE ALSO

`msgget(2)`, `msgop(2)`.

**NAME**

`msgget` — get message queue

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget (key, msgflg)
key_t key;
int msgflg;
```

**DESCRIPTION**

*Msgget* returns the message queue identifier associated with *key*.

A message queue identifier and associated message queue and data structure (see *intro(2)*) are created for *key* if one of the following are true:

*Key* is equal to `IPC_PRIVATE`.

*Key* does not already have a message queue identifier associated with it, and (*msgflg* & `IPC_CREAT`) is “true”.

Upon creation, the data structure associated with the new message queue identifier is initialized as follows:

`Msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid`, and `msg_perm.gid` are set equal to the effective user ID and effective group ID, respectively, of the calling process.

The low-order 9 bits of `msg_perm.mode` are set equal to the low-order 9 bits of *msgflg*.

`Msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime`, and `msg_rtime` are set equal to 0.

`Msg_ctime` is set equal to the current time.

`Msg_qbytes` is set equal to the system limit.

*Msgget* will fail if one or more of the following are true:

A message queue identifier exists for *key* but operation permission (see *intro(2)*) as specified by the low-order 9 bits of *msgflg* would not be granted. [EACCES]

A message queue identifier does not exist for *key* and (*msgflg* & `IPC_CREAT`) is “false”. [ENOENT]

A message queue identifier is to be created but the system imposed limit on the maximum number of allowed message queue identifiers system wide would be exceeded. [ENOSPC]

A message queue identifier exists for *key* but (*msgflg* & `IPC_CREAT`) & (*msgflg* & `IPC_EXCL`) is “true”. [EEXIST]

**RETURN VALUE**

Upon successful completion, a non-negative integer, namely a message queue identifier is returned. Otherwise, a value of `-1` is returned and *errno* is set to indicate the error.

**SEE ALSO**

`msgctl(2)`, `msgop(2)`.

## NAME

msgop — message operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd (msqid, msgp, msgsz, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz, msgflg;

int msgrcv (msqid, msgp, msgsz, msgtyp, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz;
long msgtyp;
int msgflg;
```

## DESCRIPTION

Msgsnd is used to send a message to the queue associated with the message queue identifier specified by *msqid*. {WRITE} *Msgp* points to a structure containing the message. This structure is composed of the following members:

```
long    mtype;    /* message type */
char    mtext[]; /* message text */
```

*Mtype* is a positive integer that can be used by the receiving process for message selection (see *msgrcv* below). *Mtext* is any text of length *msgsz* bytes. *Msgsz* can range from 0 to a system imposed maximum.

*Msgflg* specifies the action to be taken if one or more of the following are true:

The number of bytes already on the queue is equal to *msg\_qbytes* (see *intro(2)*).

The total number of messages on all queues system wide is equal to the system imposed limit.

These actions are as follows:

If (*msgflg* & IPC\_NOWAIT) is “true”, the message will not be sent and the calling process will return immediately.

If (*msgflg* & IPC\_NOWAIT) is “false”, the calling process will suspend execution until one of the following occurs:

The condition responsible for the suspension no longer exists, in which case the message is sent.

*Msqid* is removed from the system (see *msgctl(2)*). When this occurs, *errno* is set equal to EIDRM, and a value of -1 is returned.

The calling process receives a signal that is to be caught. In this case the message is not sent and the calling process resumes execution in the manner prescribed in *signal(2)*.

*Msgsnd* will fail and no message will be sent if one or more of the following are true:

*Msqid* is not a valid message queue identifier. [EINVAL]

Operation permission is denied to the calling process (see *intro(2)*). [EACCES]

*Mtype* is less than 1. [EINVAL]

The message cannot be sent for one of the reasons cited above and (*msgflg* & *IPC\_NOWAIT*) is "true". [EAGAIN]

*Msgsz* is less than zero or greater than the system imposed limit. [EINVAL]

*Msgp* points to an illegal address. [EFAULT]

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see *intro(2)*).

*Msg\_qnum* is incremented by 1.

*Msg\_lspid* is set equal to the process ID of the calling process.

*Msg\_stime* is set equal to the current time.

*Msgrcv* reads a message from the queue associated with the message queue identifier specified by *msqid* and places it in the structure pointed to by *msgp*. {READ} This structure is composed of the following members:

```
long    mtype;      /* message type */
char    mtext[];   /* message text */
```

*Mtype* is the received message's type as specified by the sending process. *Mtext* is the text of the message. *Msgsz* specifies the size in bytes of *mtext*. The received message is truncated to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & *MSG\_NOERROR*) is "true". The truncated part of the message is lost and no indication of the truncation is given to the calling process.

*Msgtyp* specifies the type of message requested as follows:

If *msgtyp* is equal to 0, the first message on the queue is received.

If *msgtyp* is greater than 0, the first message of type *msgtyp* is received.

If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* is received.

*Msgflg* specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

If (*msgflg* & *IPC\_NOWAIT*) is "true", the calling process will return immediately with a return value of -1 and *errno* set to *ENOMSG*.

If (*msgflg* & *IPC\_NOWAIT*) is "false", the calling process will suspend execution until one of the following occurs:

A message of the desired type is placed on the queue.

*Msqid* is removed from the system. When this occurs, *errno* is set equal to *EIDRM*, and a value of -1 is returned.

The calling process receives a signal that is to be caught. In this case a message is not received and the calling process resumes execution in the manner prescribed in *signal(2)*.

*Msgrcv* will fail and no message will be received if one or more of the following are true:

*Msqid* is not a valid message queue identifier. [EINVAL]

Operation permission is denied to the calling process. [EACCES]

*Msgsz* is less than 0. [EINVAL]

*Mtext* is greater than *msgsz* and (*msgflg* & MSG\_NOERROR) is "false". [E2BIG]

The queue does not contain a message of the desired type and (*msgtyp* & IPC\_NOWAIT) is "true". [ENOMSG]

*Msgp* points to an illegal address. [EFAULT]

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see intro (2)).

*Msg\_qnum* is decremented by 1.

*Msg\_lrpid* is set equal to the process ID of the calling process.

*Msg\_rtime* is set equal to the current time.

#### RETURN VALUES

If *msgsnd* or *msgrcv* return due to the receipt of a signal, a value of  $-1$  is returned to the calling process and *errno* is set to EINTR. If they return due to removal of *msqid* from the system, a value of  $-1$  is returned and *errno* is set to EIDRM.

Upon successful completion, the return value is as follows:

*Msgsnd* returns a value of 0.

*Msgrcv* returns a value equal to the number of bytes actually placed into *mtext*.

Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

#### SEE ALSO

*msgctl*(2), *msgget*(2).

**NAME**

*nice* — change priority of a process

**SYNOPSIS**

```
int nice (incr)
int incr;
```

**DESCRIPTION**

*Nice* adds the value of *incr* to the nice value of the calling process. A process's *nice value* is a positive number for which a more positive value results in lower CPU priority.

A maximum nice value of 39 and a minimum nice value of 0 are imposed by the system. Requests for values above or below these limits result in the nice value being set to the corresponding limit.

*Nice* will fail and not change the nice value if *incr* is negative and the effective user ID of the calling process is not super-user. [EPERM]

**RETURN VALUE**

Upon successful completion, *nice* returns the new nice value minus 20. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

*nice*(1), *exec*(2).



## NAME

`open` — open for reading or writing

## SYNOPSIS

```
#include <fcntl.h>
int open (path, oflag [ , mode ] )
char *path;
int oflag, mode;
```

## DESCRIPTION

*Path* points to a path name naming a file. *Open* opens a file descriptor for the named file and sets the file status flags according to the value of *oflag*. *Oflag* values are constructed by or-ing flags from the following list (only one of the first three flags below may be used):

**O\_RDONLY** Open for reading only.  
**O\_WRONLY** Open for writing only.  
**O\_RDWR** Open for reading and writing.  
**O\_NDELAY** This flag may affect subsequent reads and writes. See *read(2)* and *write(2)*.

When opening a FIFO with **O\_RDONLY** or **O\_WRONLY** set:

If **O\_NDELAY** is set:

An *open* for reading-only will return without delay. An *open* for writing-only will return an error if no process currently has the file open for reading.

If **O\_NDELAY** is clear:

An *open* for reading-only will block until a process opens the file for writing. An *open* for writing-only will block until a process opens the file for reading.

When opening a file associated with a communication line:

If **O\_NDELAY** is set:

The open will return without waiting for carrier.

If **O\_NDELAY** is clear:

The open will block until carrier is present.

**O\_APPEND** If set, the file pointer will be set to the end of the file prior to each write.

**O\_CREAT** If the file exists, this flag has no effect. Otherwise, the file's owner ID is set to the process's effective user ID, the file's group ID is set to the process's effective group ID, and the low-order 12 bits of the file mode are set to the value of *mode* modified as follows (see *creat(2)*):

All bits set in the process's file mode creation mask are cleared. See *umask(2)*.

The "save text image after execution bit" of the mode is cleared. See *chmod(2)*.

**O\_TRUNC** If the file exists, its length is truncated to 0 and the mode and owner are unchanged.

**O\_EXCL** If **O\_EXCL** and **O\_CREAT** are set, *open* will fail if the file exists.

Upon successful completion a non-negative integer, the file descriptor, is returned.

The file pointer used to mark the current position within the file is set to the beginning of the file.

The new file descriptor is set to remain open across *exec* system calls. See *fcntl(2)*.

No process may have more than 20 file descriptors open simultaneously.

The named file is opened unless one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

O\_CREAT is not set and the named file does not exist. [ENOENT]

A component of the path prefix denies search permission. [EACCES]

*Oflag* permission is denied for the named file. [EACCES]

The named file is a directory and *oflag* is write or read/write. [EISDIR]

The named file resides on a read-only file system and *oflag* is write or read/write. [EROFS]

Twenty (20) file descriptors are currently open. [EMFILE]

The named file is a character special or block special file, and the device associated with this special file does not exist. [ENXIO]

The file is a pure procedure (shared text) file that is being executed and *oflag* is write or read/write. [ETXTBSY]

*Path* points outside the process's allocated address space. [EFAULT]

O\_CREAT and O\_EXCL are set, and the named file exists. [EEXIST]

O\_NDELAY is set, the named file is a FIFO, O\_WRONLY is set, and no process has the file open for reading. [ENXIO]

#### RETURN VALUE

Upon successful completion, a non-negative integer, namely a file descriptor, is returned. Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

#### SEE ALSO

*close(2)*, *creat(2)*, *dup(2)*, *fcntl(2)*, *lseek(2)*, *read(2)*, *write(2)*.

**NAME**

pause — suspend process until signal

**SYNOPSIS**

pause ( )

**DESCRIPTION**

*Pause* suspends the calling process until it receives a signal. The signal must be one that is not currently set to be ignored by the calling process.

If the signal causes termination of the calling process, *pause* will not return.

If the signal is *caught* by the calling process and control is returned from the signal catching-function (see *signal(2)*), the calling process resumes execution from the point of suspension; with a return value of  $-1$  from *pause* and *errno* set to *EINTR*.

**SEE ALSO**

alarm(2), kill(2), signal(2), wait(2).

**NAME**

pipe — create an interprocess channel

**SYNOPSIS**

```
int pipe (fildes)
int fildes[2];
```

**DESCRIPTION**

*Pipe* creates an I/O mechanism called a pipe and returns two file descriptors, *fildes*[0] and *fildes*[1]. *Fildes*[0] is opened for reading and *fildes*[1] is opened for writing.

Writes up to 5120 bytes of data are buffered by the pipe before the writing process is blocked. A read on file descriptor *fildes*[0] accesses the data written to *fildes*[1] on a first-in-first-out basis.

No process may have more than 20 file descriptors open simultaneously.

*Pipe* will fail if 19 or more file descriptors are currently open. [EMFILE]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

sh(1), read(2), write(2).

**NAME**

**plock** — lock process, text, or data in memory

**SYNOPSIS**

```
#include <sys/lock.h>
int plock (op)
int op;
```

**DESCRIPTION**

*Plock* allows the calling process to lock its text segment (text lock), its data segment (data lock), or both its text and data segments (process lock) into memory. Locked segments are immune to all routine swapping. *Plock* also allows these segments to be unlocked. The effective user ID of the calling process must be super-user to use this call. *Op* specifies the following:

- PROCLOCK** — lock text & data segments into memory (process lock)
- TXTLOCK** — lock text segment into memory (text lock)
- DATLOCK** — lock data segment into memory (data lock)
- UNLOCK** — remove locks

*Plock* will fail and not perform the requested operation if one or more of the following are true:

The effective user ID of the calling process is not super-user. [EPERM]

*Op* is equal to **PROCLOCK** and a process lock, a text lock, or a data lock already exists on the calling process. [EINVAL]

*Op* is equal to **TXTLOCK** and a text lock, or a process lock already exists on the calling process. [EINVAL]

*Op* is equal to **DATLOCK** and a data lock, or a process lock already exists on the calling process. [EINVAL]

*Op* is equal to **UNLOCK** and no type of lock exists on the calling process. [EINVAL]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned to the calling process. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

`exec(2)`, `exit(2)`, `fork(2)`.

**NAME**

profil — execution time profile

**SYNOPSIS**

```
void profil (buff, bufsiz, offset, scale)
char *buff;
int bufsiz, offset, scale;
```

**DESCRIPTION**

*Buff* points to an area of core whose length (in bytes) is given by *bufsiz*. After this call, the user's program counter (*pc*) is examined each clock tick (60th second); *offset* is subtracted from it, and the result multiplied by *scale*. If the resulting number corresponds to a word inside *buff*, that word is incremented.

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0177777 (octal) gives a 1-1 mapping of *pc*'s to words in *buff*; 077777 (octal) maps each pair of instruction words together. 02(8) maps all instructions onto the beginning of *buff* (producing a non-interrupting core clock).

Profiling is turned off by giving a *scale* of 0 or 1. It is rendered ineffective by giving a *bufsiz* of 0. Profiling is turned off when an *exec* is executed, but remains on in child and parent both after a *fork*. Profiling will be turned off if an update in *buff* would cause a memory fault.

**RETURN VALUE**

Not defined.

**SEE ALSO**

prof(1), monitor(3C).

## NAME

ptrace — process trace

## SYNOPSIS

```
int ptrace (request, pid, addr, data);
int request, pid, addr, data;
```

## DESCRIPTION

*Ptrace* provides a means by which a parent process may control the execution of a child process. Its primary use is for the implementation of breakpoint debugging; see *sdb*(1). The child process behaves normally until it encounters a signal (see *signal*(2) for the list), at which time it enters a stopped state and its parent is notified via *wait*(2). When the child is in the stopped state, its parent can examine and modify its “core image” using *ptrace*. Also, the parent can cause the child either to terminate or continue, with the possibility of ignoring the signal that caused it to stop.

The *request* argument determines the precise action to be taken by *ptrace* and is one of the following:

- 0 This request must be issued by the child process if it is to be traced by its parent. It turns on the child’s trace flag that stipulates that the child should be left in a stopped state upon receipt of a signal rather than the state specified by *func*; see *signal*(2). The *pid*, *addr*, and *data* arguments are ignored, and a return value is not defined for this request. Peculiar results will ensue if the parent does not expect to trace the child.

The remainder of the requests can only be used by the parent process. For each, *pid* is the process ID of the child. The child must be in a stopped state before these requests are made.

- 1, 2 With these requests, the word at location *addr* in the address space of the child is returned to the parent process. If I and D space are separated (as on PDP-11s), request 1 returns a word from I space, and request 2 returns a word from D space. If I and D space are not separated (as on the 3B-20 and VAX-11/780), either request 1 or request 2 may be used with equal results. The *data* argument is ignored. These two requests will fail if *addr* is not the start address of a word, in which case a value of -1 is returned to the parent process and the parent’s *errno* is set to EIO.
- 3 With this request, the word at location *addr* in the child’s USER area in the system’s address space (see <sys/user.h>) is returned to the parent process. Addresses in this area range from 0 to 1024 on the PDP-11s and 0 to 2048 on the 3B-20 and VAX. The *data* argument is ignored. This request will fail if *addr* is not the start address of a word or is outside the USER area, in which case a value of -1 is returned to the parent process and the parent’s *errno* is set to EIO.
- 4, 5 With these requests, the value given by the *data* argument is written into the address space of the child at location *addr*. If I and D space are separated (as on PDP-11s), request 4 writes a word into I space, and request 5 writes a word into D space. If I and D space are not separated (as on the 3B-20 and VAX), either request 4 or request 5 may be used with equal results. Upon successful completion, the value written into the address space of the child is returned to the parent. These two requests will fail if *addr* is a location in a pure procedure

space and another process is executing in that space, or *addr* is not the start address of a word. Upon failure a value of  $-1$  is returned to the parent process and the parent's *errno* is set to EIO.

- 6 With this request, a few entries in the child's USER area can be written. *Data* gives the value that is to be written and *addr* is the location of the entry. The few entries that can be written are:

the general registers (i.e., registers 0–11 on the 3B-20, registers 0–7 on PDP-11s, and registers 0–15 on the VAX)

the condition codes of the Processor Status Word on the 3B-20.

the floating point status register and six floating point registers on PDP-11s

certain bits of the Processor Status Word on PDP-11s (i.e., bits 0–4, and 8–11)

certain bits of the Processor Status Longword on the VAX (i.e., bits 0–7, 16–20, and 30–31)

- 7 This request causes the child to resume execution. If the *data* argument is 0, all pending signals including the one that caused the child to stop are canceled before it resumes execution. If the *data* argument is a valid signal number, the child resumes execution as if it had incurred that signal and any other pending signals are canceled. The *addr* argument must be equal to 1 for this request. Upon successful completion, the value of *data* is returned to the parent. This request will fail if *data* is not 0 or a valid signal number, in which case a value of  $-1$  is returned to the parent process and the parent's *errno* is set to EIO.

- 8 This request causes the child to terminate with the same consequences as *exit(2)*.

- 9 This request sets the trace bit in the Processor Status Word of the child (i.e., bit 4 on PDP-11s; bit 30 on the VAX) and then executes the same steps as listed above for request 7. The trace bit causes an interrupt upon completion of one machine instruction. This effectively allows single stepping of the child. On the 3B-20 there is no trace bit and this request returns an error.

Note: the trace bit remains set after an interrupt on PDP-11s but is turned off after an interrupt on the VAX.

To forestall possible fraud, *ptrace* inhibits the set-user-id facility on subsequent *exec(2)* calls. If a traced process calls *exec*, it will stop before executing the first instruction of the new image showing signal SIGTRAP.

#### GENERAL ERRORS

*Ptrace* will in general fail if one or more of the following are true:

*Request* is an illegal number. [EIO]

*Pid* identifies a child that does not exist or has not executed a *ptrace* with request 0. [ESRCH]

#### SEE ALSO

sdb(1), exec(2), signal(2), wait(2).



**NAME**

read — read from file

**SYNOPSIS**

```
int read (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

**DESCRIPTION**

*Fildes* is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call.

*Read* attempts to read *nbyte* bytes from the file associated with *fildes* into the buffer pointed to by *buf*.

On devices capable of seeking, the *read* starts at a position in the file given by the file pointer associated with *fildes*. Upon return from *read*, the file pointer is incremented by the number of bytes actually read.

Devices that are incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

Upon successful completion, *read* returns the number of bytes actually read and placed in the buffer; this number may be less than *nbyte* if the file is associated with a communication line (see *ioctl(2)* and *termio(7)*), or if the number of bytes left in the file is less than *nbyte* bytes. A value of 0 is returned when an end-of-file has been reached.

When attempting to read from an empty pipe (or FIFO):

If *O\_NDELAY* is set, the read will return a 0.

If *O\_NDELAY* is clear, the read will block until data is written to the file or the file is no longer open for writing.

When attempting to read a file associated with a tty that has no data currently available:

If *O\_NDELAY* is set, the read will return a 0.

If *O\_NDELAY* is clear, the read will block until data becomes available.

*Read* will fail if one or more of the following are true:

*Fildes* is not a valid file descriptor open for reading. [EBADF]

*Buf* points outside the allocated address space. [EFAULT]

**RETURN VALUE**

Upon successful completion a non-negative integer is returned indicating the number of bytes actually read. Otherwise, a -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

*creat(2)*, *dup(2)*, *fcntl(2)*, *ioctl(2)*, *open(2)*, *pipe(2)*, *termio(7)*.



## NAME

semctl — semaphore control operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl (semid, semnum, cmd, arg)
int semid, cmd;
int semnum;
union semun {
    int val;
    struct semid_ds *buf;
    ushort array[ ];
} arg;
```

## DESCRIPTION

*Semctl* provides a variety of semaphore control operations as specified by *cmd*.

The following *cmds* are executed with respect to the semaphore specified by *semid* and *semnum*:

|         |                                                                                                                                                                                                        |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GETVAL  | Return the value of <i>semval</i> (see <i>intro(2)</i> ). {READ}                                                                                                                                       |
| SETVAL  | Set the value of <i>semval</i> to <i>arg.val</i> . {ALTER} When this <i>cmd</i> is successfully executed the <i>semadj</i> value corresponding to the specified semaphore in all processes is cleared. |
| GETPID  | Return the value of <i>sempid</i> . {READ}                                                                                                                                                             |
| GETNCNT | Return the value of <i>semncnt</i> . {READ}                                                                                                                                                            |
| GETZCNT | Return the value of <i>semzcnt</i> . {READ}                                                                                                                                                            |

The following *cmds* return and set, respectively, every *semval* in the set of semaphores.

|        |                                                                                                                                                                                                                                   |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GETALL | Place <i>semvals</i> into array pointed to by <i>arg.array</i> . {READ}                                                                                                                                                           |
| SETALL | Set <i>semvals</i> according to the array pointed to by <i>arg.array</i> . {ALTER} When this <i>cmd</i> is successfully executed the <i>semadj</i> values corresponding to each specified semaphore in all processes are cleared. |

The following *cmds* are also available:

|          |                                                                                                                                                                                                                                                                        |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IPC_STAT | Place the current value of each member of the data structure associated with <i>semid</i> into the structure pointed to by <i>arg.buf</i> . The contents of this structure are defined in <i>intro(2)</i> . {READ}                                                     |
| IPC_SET  | Set the value of the following members of the data structure associated with <i>semid</i> to the corresponding value found in the structure pointed to by <i>arg.buf</i> :<br><b>sem_perm.uid</b><br><b>sem_perm.gid</b><br><b>sem_perm.mode /* only low 9 bits */</b> |

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super user or to the value of *sem\_perm.uid* in the data structure associated with *semid*.

**IPC\_RMID** Remove the semaphore identifier specified by *semid* from the system and destroy the set of semaphores and data structure associated with it. This cmd can only be executed by a process that has an effective user ID equal to either that of super user or to the value of *sem\_perm.uid* in the data structure associated with *semid*.

*Semctl* will fail if one or more of the following are true:

*Semid* is not a valid semaphore identifier. [EINVAL]

*Semnum* is less than zero or greater than *sem\_nsems*. [EINVAL]

*Cmd* is not a valid command. [EINVAL]

Operation permission is denied to the calling process (see *intro(2)*). [EACCESS]

*Cmd* is SETVAL or SETALL and the value to which *semval* is to be set is greater than the system imposed maximum. [ERANGE]

*Cmd* is equal to IPC\_RMID or IPC\_SET and the effective user ID of the calling process is not equal to that of super user and it is not equal to the value of *sem\_perm.uid* in the data structure associated with *semid*. [EPERM]

*Arg.buf* points to an illegal address. [EFAULT]

## RETURN VALUE

Upon successful completion, the value returned depends on *cmd* as follows:

|            |                               |
|------------|-------------------------------|
| GETVAL     | The value of <i>semval</i> .  |
| GETPID     | The value of <i>sempid</i> .  |
| GETNCNT    | The value of <i>semncnt</i> . |
| GETZCNT    | The value of <i>semzcnt</i> . |
| All others | A value of 0.                 |

Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

## SEE ALSO

*semget(2)*, *semop(2)*.

## NAME

semget — get set of semaphores

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget (key, nsems, semflg)
key_t key;
int nsems, semflg;
```

## DESCRIPTION

*Semget* returns the semaphore identifier associated with *key*.

A semaphore identifier and associated data structure and set containing *nsems* semaphores (see *intro(2)*) are created for *key* if one of the following are true:

*Key* is equal to `IPC_PRIVATE`.

*Key* does not already have a semaphore identifier associated with it, and  $(semflg \& IPC\_CREAT)$  is “true”.

Upon creation, the data structure associated with the new semaphore identifier is initialized as follows:

`Sem_perm.cuid`, `sem_perm.uid`, `sem_perm.cgid`, and `sem_perm.gid` are set equal to the effective user ID and effective group ID, respectively, of the calling process.

The low-order 9 bits of `sem_perm.mode` are set equal to the low-order 9 bits of *semflg*.

`Sem_nsems` is set equal to the value of *nsems*.

`Sem_otime` is set equal to 0 and `sem_ctime` is set equal to the current time.

*Semget* will fail if one or more of the following are true:

*Nsems* is either less than or equal to zero or greater than the system imposed limit. [EINVAL]

A semaphore identifier exists for *key* but operation permission (see *intro(2)*) as specified by the low-order 9 bits of *semflg* would not be granted. [EACCES]

A semaphore identifier exists for *key* but the number of semaphores in the set associated with it is less than *nsems* and *nsems* is not equal to zero. [EINVAL]

A semaphore identifier does not exist for *key* and  $(semflg \& IPC\_CREAT)$  is “false”. [ENOENT]

A semaphore identifier is to be created but the system imposed limit on the maximum number of allowed semaphore identifiers system wide would be exceeded. [ENOSPC]

A semaphore identifier is to be created but the system imposed limit on the maximum number of allowed semaphores system wide would be exceeded. [ENOSPC]

A semaphore identifier exists for *key* but  $(semflg \& IPC\_CREAT) \& (semflg \& IPC\_EXCL)$  is “true”. [EEXIST]

**RETURN VALUE**

Upon successful completion, a non-negative integer, namely a semaphore identifier is returned. Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

**SEE ALSO**

`semctl(2)`, `semop(2)`.

## NAME

semop — semaphore operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop (semid, sops, nsops)
int semid;
struct sembuf (*sops)[];
int nsops;
```

## DESCRIPTION

*Semop* is used to atomically perform an array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by *semid*. *Sops* is a pointer to the array of semaphore-operation structures. *Nsops* is the number of such structures in the array. The contents of each structure includes the following members:

```
short sem_num; /* semaphore number */
short sem_op; /* semaphore operation */
short sem_flg; /* operation flags */
```

Each semaphore operation specified by *sem\_op* is performed on the corresponding semaphore specified by *semid* and *sem\_num*.

*Sem\_op* specifies one of three semaphore operations as follows:

If *sem\_op* is a negative integer, one of the following will occur:  
{ALTER}

If *semval* (see *intro(2)*) is greater than or equal to the absolute value of *sem\_op*, the absolute value of *sem\_op* is subtracted from *semval*. Also, if (*sem\_flg* & SEM\_UNDO) is "true", the absolute value of *sem\_op* is added to the calling process's *semadj* value (see *exit(2)*) for the specified semaphore.

If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* & IPC\_NOWAIT) is "true", *semop* will return immediately.

If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* & IPC\_NOWAIT) is "false", *semop* will increment the *semncnt* associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:

*Semval* becomes greater than or equal to the absolute value of *sem\_op*. When this occurs, the value of *semncnt* associated with the specified semaphore is decremented, the absolute value of *sem\_op* is subtracted from *semval* and, if (*sem\_flg* & SEM\_UNDO) is "true", the absolute value of *sem\_op* is added to the calling process's *semadj* value for the specified semaphore.

The *semid* for which the calling process is awaiting action is removed from the system (see *semctl(2)*). When this occurs, *errno* is set equal to EIDRM, and a value of -1 is returned.

The calling process receives a signal that is to be caught. When this occurs, the value of *semncnt* associated with

the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in *signal(2)*.

If *sem\_op* is a positive integer, the value of *sem\_op* is added to *semval* and, if (*sem\_flg* & SEM\_UNDO) is "true", the value of *sem\_op* is subtracted from the calling process's *semadj* value for the specified semaphore. {ALTER}

If *sem\_op* is zero, one of the following will occur: {READ}

If *semval* is zero, *semop* will return immediately.

If *semval* is not equal to zero and (*sem\_flg* & IPC\_NOWAIT) is "true", *semop* will return immediately.

If *semval* is not equal to zero and (*sem\_flg* & IPC\_NOWAIT) is "false", *semop* will increment the *semzcnt* associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:

*semval* becomes zero, at which time the value of *semzcnt* associated with the specified semaphore is decremented.

The *semid* for which the calling process is awaiting action is removed from the system. When this occurs, *errno* is set equal to EIDRM, and a value of -1 is returned.

The calling process receives a signal that is to be caught. When this occurs, the value of *semzcnt* associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in *signal(2)*.

*Semop* will fail if one or more of the following are true for any of the semaphore operations specified by *sops*:

*Semid* is not a valid semaphore identifier. [EINVAL]

*Sem\_num* is less than zero or greater than or equal to the number of semaphores in the set associated with *semid*. [EFBIG]

*Nsops* is greater than the system imposed maximum. [E2BIG]

Operation permission is denied to the calling process (see *intro(2)*). [EACCES]

The operation would result in suspension of the calling process but (*sem\_flg* & IPC\_NOWAIT) is "true". [EAGAIN]

The limit on the number of individual processes requesting an SEM\_UNDO would be exceeded. [ENOSPC]

The number of individual semaphores for which the calling process requests a SEM\_UNDO would exceed the limit. [EINVAL]

An operation would cause a *semval* to overflow the system imposed limit. [ERANGE]

An operation would cause a *semadj* value to overflow the system imposed limit. [ERANGE]

*Sops* points to an illegal address. [EFAULT]

Upon successful completion, the value of *sempid* for each semaphore specified in the array pointed to by *sops* is set equal to the process ID of the calling process.

**RETURN VALUE**

If *semop* returns due to the receipt of a signal, a value of  $-1$  is returned to the calling process and *errno* is set to EINTR. If it returns due to the removal of a *semid* from the system, a value of  $-1$  is returned and *errno* is set to EIDRM.

Upon successful completion, the value of *semval* at the time of the call for the last operation in the array pointed to by *sops* is returned. Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

**SEE ALSO**

*exec(2)*, *exit(2)*, *fork(2)*, *semctl(2)*, *semget(2)*.



**NAME**

setpgrp — set process group ID

**SYNOPSIS**

```
int setpgrp ( )
```

**DESCRIPTION**

*Setpgrp* sets the process group ID of the calling process to the process ID of the calling process and returns the new process group ID.

**RETURN VALUE**

*Setpgrp* returns the value of the new process group ID.

**SEE ALSO**

*exec(2)*, *fork(2)*, *getpid(2)*, *intro(2)*, *kill(2)*, *signal(2)*.

**NAME**

setuid, setgid — set user and group IDs

**SYNOPSIS**

```
int setuid (uid)
int uid;

int setgid (gid)
int gid;
```

**DESCRIPTION**

*Setuid (setgid)* is used to set the real user (group) ID and effective user (group) ID of the calling process.

If the effective user ID of the calling process is super-user, the real user (group) ID and effective user (group) ID are set to *uid (gid)*.

If the effective user ID of the calling process is not super-user, but its real user (group) ID is equal to *uid (gid)*, the effective user (group) ID is set to *uid (gid)*.

*Setuid (setgid)* will fail if the real user (group) ID of the calling process is not equal to *uid (gid)* and its effective user ID is not super-user. [EPERM]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

getuid(2), intro(2).

## NAME

shmctl — shared memory control operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl (shmid, cmd, buf)
int shmid, cmd;
struct shmids *buf;
```

## DESCRIPTION

*Shmctl* provides a variety of shared memory control operations as specified by *cmd*. The following *cmds* are available:

- IPC\_STAT** Place the current value of each member of the data structure associated with *shmid* into the structure pointed to by *buf*. The contents of this structure are defined in *intro(2)*. {READ}
- IPC\_SET** Set the value of the following members of the data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:
- ```
shm_perm.uid
shm_perm.gid
shm_perm.mode /* only low 9 bits */
```
- This *cmd* can only be executed by a process that has an effective user ID equal to either that of super user or to the value of *shm\_perm.uid* in the data structure associated with *shmid*.
- IPC\_RMID** Remove the shared memory identifier specified by *shmid* from the system and destroy the shared memory segment and data structure associated with it. This *cmd* can only be executed by a process that has an effective user ID equal to either that of super user or to the value of *shm\_perm.uid* in the data structure associated with *shmid*.

*Shmctl* will fail if one or more of the following are true:

*Shmid* is not a valid shared memory identifier. [EINVAL]

*Cmd* is not a valid command. [EINVAL]

*Cmd* is equal to *IPC\_STAT* and {READ} operation permission is denied to the calling process (see *intro(2)*). [EACCESS]

*Cmd* is equal to *IPC\_RMID* or *IPC\_SET* and the effective user ID of the calling process is not equal to that of super user and it is not equal to the value of *shm\_perm.uid* in the data structure associated with *shmid*. [EPERM]

*Buf* points to an illegal address. [EFAULT]

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## SEE ALSO

shmget(2), shmop(2).

## NAME

shmget — get shared memory segment

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget (key, size, shmflg)
key_t key;
int size, shmflg;
```

## DESCRIPTION

*Shmget* returns the shared memory identifier associated with *key*.

A shared memory identifier and associated data structure and shared memory segment of size *size* bytes (see *intro(2)*) are created for *key* if one of the following are true:

*Key* is equal to `IPC_PRIVATE`.

*Key* does not already have a shared memory identifier associated with it, and (*shmflg* & `IPC_CREAT`) is “true”.

Upon creation, the data structure associated with the new shared memory identifier is initialized as follows:

`Shm_perm.cuid`, `shm_perm.uid`, `shm_perm.cgid`, and `shm_perm.gid` are set equal to the effective user ID and effective group ID, respectively, of the calling process.

The low-order 9 bits of `shm_perm.mode` are set equal to the low-order 9 bits of *shmflg*. `Shm_segsz` is set equal to the value of *size*.

`Shm_lpid`, `shm_nattch`, `shm_atime`, and `shm_dtime` are set equal to 0.

`Shm_ctime` is set equal to the current time.

*Shmget* will fail if one or more of the following are true:

*Size* is less than the system imposed minimum or greater than the system imposed maximum. [EINVAL]

A shared memory identifier exists for *key* but operation permission (see *intro(2)*) as specified by the low-order 9 bits of *shmflg* would not be granted. [EACCES]

A shared memory identifier exists for *key* but the size of the segment associated with it is less than *size* and *size* is not equal to zero. [EINVAL]

A shared memory identifier does not exist for *key* and (*shmflg* & `IPC_CREAT`) is “false”. [ENOENT]

A shared memory identifier is to be created but the system imposed limit on the maximum number of allowed shared memory identifiers system wide would be exceeded. [ENOSPC]

A shared memory identifier and associated shared memory segment are to be created but the amount of available physical memory is not sufficient to fill the request. [ENOMEM]

A shared memory identifier exists for *key* but ( (*shmflg* & `IPC_CREAT`) & ( *shmflg* & `IPC_EXCL` ) ) is “true”. [EEXIST]

**RETURN VALUE**

Upon successful completion, a non-negative integer, namely a shared memory identifier is returned. Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

**SEE ALSO**

shmctl(2), shmop(2).

## NAME

shmop — shared memory operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

char *shmat (shmid, shmaddr, shmflg)
int shmid;
char *shmaddr
int shmflg;

int shmdt (shmaddr)
char *shmaddr
```

## DESCRIPTION

*Shmat* attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the data segment of the calling process. The segment is attached at the address specified by one of the following criteria:

If *shmaddr* is equal to zero, the segment is attached at the first available address as selected by the system.

If *shmaddr* is not equal to zero and (*shmflg* & SHM\_RND) is “true”, the segment is attached at the address given by (*shmaddr* - (*shmaddr* modulus SHMLBA)).

If *shmaddr* is not equal to zero and (*shmflg* & SHM\_RND) is “false”, the segment is attached at the address given by *shmaddr*.

The segment is attached for reading if (*shmflg* & SHM\_RDONLY) is “true” {READ}, otherwise it is attached for reading and writing {READ/WRITE}.

*Shmat* will fail and not attach the shared memory segment if one or more of the following are true:

*Shmid* is not a valid shared memory identifier. [EINVAL]

Operation permission is denied to the calling process (see *intro(2)*). [EACCESS]

The available data space is not large enough to accommodate the shared memory segment. [ENOMEM]

*Shmaddr* is not equal to zero, and the value of (*shmaddr* - (*shmaddr* modulus SHMLBA)) is an illegal address. [EINVAL]

*Shmaddr* is not equal to zero, (*shmflg* & SHM\_RND) is “false”, and the value of *shmaddr* is an illegal address. [EINVAL]

The number of shared memory segments attached to the calling process would exceed the system imposed limit. [EMFILE]

*Shmdt* detaches from the calling process’s data segment the shared memory segment located at the address specified by *shmaddr*.

*Shmdt* will fail and not detach the shared memory segment if *shmaddr* is not the data segment start address of a shared memory segment. [EINVAL]

## RETURN VALUES

Upon successful completion, the return value is as follows:

*Shmat* returns the data segment start address of the attached shared memory segment.

*Shmctl* returns a value of 0.

Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

**SEE ALSO**

*exec(2)*, *exit(2)*, *fork(2)*, *shmctl(2)*, *shmget(2)*.

**NAME**

signal — specify what to do upon receipt of a signal

**SYNOPSIS**

```
#include <sys/signal.h>

int (*signal (sig, func))()
int sig;
int (*func)();
```

**DESCRIPTION**

*Signal* allows the calling process to choose one of three ways in which it is possible to handle the receipt of a specific signal. *Sig* specifies the signal and *func* specifies the choice.

*Sig* can be assigned any one of the following except SIGKILL:

SIGHUP	01	hangup
SIGINT	02	interrupt
SIGQUIT	03*	quit
SIGILL	04*	illegal instruction (not reset when caught)
SIGTRAP	05*	trace trap (not reset when caught)
SIGIOT	06*	IOT instruction
SIGEMT	07*	EMT instruction
SIGFPE	08*	floating point exception
SIGKILL	09	kill (cannot be caught or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGUSR1	16	user defined signal 1
SIGUSR2	17	user defined signal 2
SIGCLD	18	death of a child (see <i>WARNING</i> below)
SIGPWR	19	power fail (see <i>WARNING</i> below)

See below for the significance of the asterisk (\*) in the above list.

*Func* is assigned one of three values: SIG\_DFL, SIG\_IGN, or a *function address*. The actions prescribed by these values of are as follows:

**SIG\_DFL** — terminate process upon receipt of a signal

Upon receipt of the signal *sig*, the receiving process is to be terminated with all of the consequences outlined in *exit(2)* plus a “core image” will be made in the current working directory of the receiving process if *sig* is one for which an asterisk appears in the above list *and* the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

An ordinary file named **core** exists and is writable or can be created. If the file must be created, it will have the following properties:

a mode of 0666 modified by the file creation mask (see *umask(2)*)

a file owner ID that is the same as the effective user ID of the receiving process

a file group ID that is the same as the effective group ID of the receiving process



**SIG\_IGN** — ignore signal

The signal *sig* is to be ignored.

Note: the signal SIGKILL cannot be ignored.

**function address** — catch signal

Upon receipt of the signal *sig*, the receiving process is to execute the signal-catching function pointed to by *func*. The signal number *sig* will be passed as the only argument to the signal-catching function. Before entering the signal-catching function, the value of *func* for the caught signal will be set to SIG\_DFL unless the signal is SIGILL, SIGTRAP, or SIGPWR.

Upon return from the signal-catching function, the receiving process will resume execution at the point it was interrupted.

When a signal that is to be caught occurs during a *read*, a *write*, an *open*, or an *ioctl* system call on a slow device (like a terminal; but not a file), during a *pause* system call, or during a *wait* system call that does not return immediately due to the existence of a previously stopped or zombie process, the signal catching function will be executed and then the interrupted system call will return a -1 to the calling process with *errno* set to EINTR.

Note: the signal SIGKILL cannot be caught.

A call to *signal* cancels a pending signal *sig* except for a pending SIGKILL signal.

*Signal* will fail if one or more of the following are true:

*Sig* is an illegal signal number, including SIGKILL. [EINVAL]

*Func* points to an illegal address. [EFAULT]

**RETURN VALUE**

Upon successful completion, *signal* returns the previous value of *func* for the specified signal *sig*. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

kill(1), kill(2), pause(2), ptrace(2), wait(2), setjmp(3C).

**WARNING**

Two other signals that behave differently than the signals described above exist in this release of the system; they are:

<b>SIGCLD</b>	18	death of a child (reset when caught)
<b>SIGPWR</b>	19	power fail (not reset when caught)

There is no guarantee that, in future releases of UNIX, these signals will continue to behave as described below; they are included only for compatibility with other versions of UNIX. Their use in new programs is strongly discouraged.

For these signals, *func* is assigned one of three values: SIG\_DFL, SIG\_IGN, or a *function address*. The actions prescribed by these values of are as follows:

**SIG\_DFL** - ignore signal

The signal is to be ignored.

**SIG\_IGN** - ignore signal

The signal is to be ignored. Also, if *sig* is SIGCLD, the calling process's child processes will not create zombie processes when they terminate; see *exit*(2).

*function address* - catch signal

If the signal is SIGPWR, the action to be taken is the same as that described above for *func* equal to *function address*. The same is true if the signal is SIGCLD except, that while the process is executing the signal-catching function any received SIGCLD signals will be queued and the signal-catching function will be continually reentered until the queue is empty.

The SIGCLD affects two other system calls (*wait(2)*, and *exit(2)*) in the following ways:

- wait* If the *func* value of SIGCLD is set to SIG\_IGN and a *wait* is executed, the *wait* will block until all of the calling process's child processes terminate; it will then return a value of -1 with *errno* set to ECHILD.
- exit* If in the exiting process's parent process the *func* value of SIGCLD is set to SIG\_IGN, the exiting process will not create a zombie process.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the proceeding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should take care not to set SIGCLD to be caught.

## NAME

stat, fstat — get file status

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int stat (path, buf)
char *path;
struct stat *buf;

int fstat (fildes, buf)
int fildes;
struct stat *buf;
```

## DESCRIPTION

*Path* points to a path name naming a file. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable. *Stat* obtains information about the named file.

Similarly, *fstat* obtains information about an open file known by the file descriptor *fildes*, obtained from a successful *open*, *creat*, *dup*, *fcntl*, or *pipe* system call.

*Buf* is a pointer to a *stat* structure into which information is placed concerning the file.

The contents of the structure pointed to by *buf* include the following members:

```
ushort  st_mode;    /* File mode; see mknod(2) */
ino_t   st_ino;    /* Inode number */
dev_t   st_dev;    /* ID of device containing */
                /* a directory entry for this file */
dev_t   st_rdev;   /* ID of device */
                /* This entry is defined only for */
                /* character special or block special files */
short   st_nlink;  /* Number of links */
ushort  st_uid;    /* User ID of the file's owner */
ushort  st_gid;    /* Group ID of the file's group */
off_t   st_size;   /* File size in bytes */
time_t  st_atime;  /* Time of last access */
time_t  st_mtime;  /* Time of last data modification */
time_t  st_ctime;  /* Time of last file status change */
                /* Times measured in seconds since */
                /* 00:00:00 GMT, Jan. 1, 1970 */
```

**st\_atime** Time when file data was last accessed. Changed by the following system calls: *creat(2)*, *mknod(2)*, *pipe(2)*, *utime(2)*, and *read(2)*.

**st\_mtime** Time when data was last modified. Changed by the following system calls: *creat(2)*, *mknod(2)*, *pipe(2)*, *utime(2)*, and *write(2)*.

**st\_ctime** Time when file status was last changed. Changed by the following system calls: *chmod(2)*, *chown(2)*, *creat(2)*, *link(2)*, *mknod(2)*, *pipe(2)*, *unlink(2)*, *utime(2)*, and *write(2)*.

*Stat* will fail if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied for a component of the path prefix.  
[EACCES]

*Buf* or *path* points to an invalid address. [EFAULT]

*Fstat* will fail if one or more of the following are true:

*Fildes* is not a valid open file descriptor. [EBADF]

*Buf* points to an invalid address. [EFAULT]

#### RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

#### SEE ALSO

chmod(2), chown(2), creat(2), link(2), mknod(2), time(2), unlink(2).

**NAME**

stime — set time

**SYNOPSIS**

```
int stime (tp)
long *tp;
```

**DESCRIPTION**

*Stime* sets the system's idea of the time and date. *Tp* points to the value of time as measured in seconds from 00:00:00 GMT January 1, 1970.

*Stime* will fail if the effective user ID of the calling process is not super-user. [EPERM]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

time(2).

**NAME**

`sync` — update super-block

**SYNOPSIS**

`void sync ( )`

**DESCRIPTION**

*Sync* causes all information in memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

It should be used by programs which examine a file system, for example *fsck*, *df*, etc. It is mandatory before a boot.

The writing, although scheduled, is not necessarily complete upon return from *sync*.

**NAME**

sys3b — 3B20S specific system calls

**SYNOPSIS**

```
void sys3b (cmd, arg1[, arg2])
int cmd, arg1, arg2;
```

**DESCRIPTION**

This system call provides for 3B20S specific actions. Most require super-user privileges as the effects can be dangerous. The *cmd* values available are:

- 1 Reboot the processor. This call causes an immediate entry into the bootstrap code.
- 2 System *printf* interface. *Arg1* is taken as a pointer to a null terminated string to be copied into the operating system circular print buffer.
- 3 Attach to an address translation buffer.
- 4 System namelist interface. The value of *arg1* is used to return the address of various data elements in the system.
- 5 Override for system Maintenance Reset Function (MRF) action. If *arg1* is non-zero, it is taken as the indicator for handling a processor MRF. If zero, the current setting is returned.
- 6 Send a Processor Recovery Message (PRM). *Arg1* is used as a pointer to a 16 byte string to be converted to a PRM and transmitted to the Emergency Action Interface (EAI).
- 7 Modify the System Status Register (SSR). Bits set in *arg1* are set or cleared in the SSR if *arg2* is non-zero or zero, respectively.
- 8 Read EAI Input Parameter Buffer. *Arg1* is used as a location in user space where the current Input Parameter Buffer is to be placed.
- 9 Change default Field Test Set utility-id. 10 Change the floating point flag bits in the extended processor status word.

**SEE ALSO**

fts(1M), ipb(1M), prm(1M), reboot(1M), setmrf(1M), ssr(1M), in the *UNIX System Administrator's Manual*.

**NAME**

time — get time

**SYNOPSIS**

**long time ((long \*) 0)**

**long time (tloc)**

**long \*tloc;**

**DESCRIPTION**

*Time* returns the value of time in seconds since 00:00:00 GMT, January 1, 1970.

If *tloc* (taken as an integer) is non-zero, the return value is also stored in the location to which *tloc* points.

*Time* will fail if *tloc* points to an illegal address. [EFAULT]

**RETURN VALUE**

Upon successful completion, *time* returns the value of time. Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

**SEE ALSO**

stime(2).



**NAME**

*times* — get process and child process times

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/times.h>

long times (buffer)
struct tms *buffer;
```

**DESCRIPTION**

*Times* fills the structure pointed to by *buffer* with time-accounting information. The following is this contents of the structure:

```
struct tms {
    time_t tms_utime;
    time_t tms_stime;
    time_t tms_cutime;
    time_t tms_cstime;
};
```

This information comes from the calling process and each of its terminated child processes for which it has executed a *wait*. All times are in 60ths of a second on DEC processors, 100ths of a second on WECo processors.

*Tms\_utime* is the CPU time used while executing instructions in the user space of the calling process.

*Tms\_stime* is the CPU time used by the system on behalf of the calling process.

*Tms\_cutime* is the sum of the *tms\_utimes* and *tms\_cutimes* of the child processes.

*Tms\_cstime* is the sum of the *tms\_stimes* and *tms\_cstimes* of the child processes.

*Times* will fail if *buffer* points to an illegal address. [EFAULT]

**RETURN VALUE**

Upon successful completion, *times* returns the elapsed real time, in 60ths (100ths) of a second, since an arbitrary point in the past (e.g., system start-up time). This point does not change from one invocation of *times* to another. If *times* fails, a  $-1$  is returned and *errno* is set to indicate the error.

**SEE ALSO**

*exec(2)*, *fork(2)*, *time(2)*, *wait(2)*.

**NAME**

ulimit — get and set user limits

**SYNOPSIS**

```
long ulimit (cmd, newlimit)
int cmd;
long newlimit;
```

**DESCRIPTION**

This function provides for control over process limits. The *cmd* values available are:

- 1 Get the process's file size limit. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read.
- 2 Set the process's file size limit to the value of *newlimit*. Any process may decrease this limit, but only a process with an effective user ID of super-user may increase the limit. *Ulimit* will fail and the limit will be unchanged if a process with an effective user ID other than super-user attempts to increase its file size limit. [EPERM]
- 3 Get the maximum possible break value. See *brk(2)*.

**RETURN VALUE**

Upon successful completion, a non-negative value is returned. Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

**SEE ALSO**

*brk(2)*, *write(2)*.

**NAME**

umask — set and get file creation mask

**SYNOPSIS**

```
int umask (cmask)
int cmask;
```

**DESCRIPTION**

*Umask* sets the process's file mode creation mask to *cmask* and returns the previous value of the mask. Only the low-order 9 bits of *cmask* and the file mode creation mask are used.

**RETURN VALUE**

The previous value of the file mode creation mask is returned.

**SEE ALSO**

mkdir(1), sh(1), chmod(2), creat(2), mknod(2), open(2).

**NAME**

umount — unmount a file system

**SYNOPSIS**

```
int umount (spec)
char *spec;
```

**DESCRIPTION**

*Umount* requests that a previously mounted file system contained on the block special device identified by *spec* be unmounted. *Spec* is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.

*Umount* may be invoked only by the super-user.

*Umount* will fail if one or more of the following are true:

The process's effective user ID is not super-user. [EPERM]

*Spec* does not exist. [ENXIO]

*Spec* is not a block special device. [ENOTBLK]

*Spec* is not mounted. [EINVAL]

A file on *spec* is busy. [EBUSY]

*Spec* points outside the process's allocated address space. [EFAULT]

**RETURN VALUE**

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

mount(2).

**NAME**

`uname` — get name of current UNIX system

**SYNOPSIS**

```
#include <sys/utsname.h>
```

```
int uname (name)
```

```
struct utsname *name;
```

**DESCRIPTION**

*Uname* stores information identifying the current UNIX system in the structure pointed to by *name*.

*Uname* uses the structure defined in `<sys/utsname.h>` whose members are:

```
char  sysname[9];
char  nodename[9];
char  release[9];
char  version[9];
char  machine[9];
```

*Uname* returns a null-terminated character string naming the current UNIX system in the character array *sysname*. Similarly, *nodename* contains the name that the system is known by on a communications network. *Release* and *version* further identify the operating system. *Machine* contains a standard name that identifies the hardware that UNIX is running on.

*Uname* will fail if *name* points to an invalid address. [EFAULT]

**RETURN VALUE**

Upon successful completion, a non-negative value is returned. Otherwise, `-1` is returned and *errno* is set to indicate the error.

**SEE ALSO**

`uname(1)`.

**NAME**

`unlink` — remove directory entry

**SYNOPSIS**

```
int unlink (path)
char *path;
```

**DESCRIPTION**

*Unlink* removes the directory entry named by the path name pointed to be *path*.

The named file is unlinked unless one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied for a component of the path prefix. [EACCES]

Write permission is denied on the directory containing the link to be removed. [EACCES]

The named file is a directory and the effective user ID of the process is not super-user. [EPERM]

The entry to be unlinked is the mount point for a mounted file system. [EBUSY]

The entry to be unlinked is the last link to a pure procedure (shared text) file that is being executed. [ETXTBSY]

The directory entry to be unlinked is part of a read-only file system. [EROFS]

*Path* points outside the process's allocated address space. [EFAULT]

When all links to a file have been removed and no process has the file open, the space occupied by the file is freed and the file ceases to exist. If one or more processes have the file open when the last link is removed, the removal is postponed until all references to the file have been closed.

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

`rm(1)`, `close(2)`, `link(2)`, `open(2)`.

**NAME**

ustat — get file system statistics

**SYNOPSIS**

```
#include <sys/types.h>
#include <ustat.h>
```

```
int ustat (dev, buf)
int dev;
struct ustat *buf;
```

**DESCRIPTION**

*Ustat* returns information about a mounted file system. *Dev* is a device number identifying a device containing a mounted file system. *Buf* is a pointer to a *ustat* structure that includes the following elements:

```
daddr_t f_tfree;      /* Total free blocks */
ino_t   f_tinode;    /* Number of free inodes */
char    f_fname[6];  /* Filsys name */
char    f_fpack[6];  /* Filsys pack name */
```

*Ustat* will fail if one or more of the following are true:

*Dev* is not the device number of a device containing a mounted file system. [EINVAL]

*Buf* points outside the process's allocated address space. [EFAULT]

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

**SEE ALSO**

stat(2), fs(4).

## NAME

*utime* — set file access and modification times

## SYNOPSIS

```
#include <sys/types.h>
int utime (path, times)
char *path;
struct utimbuf *times;
```

## DESCRIPTION

*Path* points to a path name naming a file. *Utime* sets the access and modification times of the named file.

If *times* is NULL, the access and modification times of the file are set to the current time. A process must be the owner of the file or have write permission to use *utime* in this manner.

If *times* is not NULL, *times* is interpreted as a pointer to a *utimbuf* structure and the access and modification times are set to the values contained in the designated structure. Only the owner of the file or the super-user may use *utime* this way.

The times in the following structure are measured in seconds since 00:00:00 GMT, Jan. 1, 1970.

```
struct utimbuf {
    time_t actime; /* access time */
    time_t modtime; /* modification time */
};
```

*Utime* will fail if one or more of the following are true:

The named file does not exist. [ENOENT]

A component of the path prefix is not a directory. [ENOTDIR]

Search permission is denied by a component of the path prefix. [EACCES]

The effective user ID is not super-user and not the owner of the file and *times* is not NULL. [EPERM]

The effective user ID is not super-user and not the owner of the file and *times* is NULL and write access is denied. [EACCES]

The file system containing the file is mounted read-only. [EROFS]

*Times* is not NULL and points outside the process's allocated address space. [EFAULT]

*Path* points outside the process's allocated address space. [EFAULT]

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## SEE ALSO

stat(2).



**NAME**

wait — wait for child process to stop or terminate

**SYNOPSIS**

```
int wait (stat_loc)
```

```
int *stat_loc;
```

```
int wait ((int *)0)
```

**DESCRIPTION**

*Wait* suspends the calling process until it receives a signal that is to be caught (see *signal(2)*), or until any one of the calling process's child processes stops in a trace mode (see *ptrace(2)*) or terminates. If a child process stopped or terminated prior to the call on *wait*, return is immediate.

If *stat\_loc* (taken as an integer) is non-zero, 16 bits of information called status are stored in the low order 16 bits of the location pointed to by *stat\_loc*. Status can be used to differentiate between stopped and terminated child processes and if the child process terminated, status identifies the cause of termination and pass useful information to the parent. This is accomplished in the following manner:

If the child process stopped, the high order 8 bits of status will contain the number of the signal that caused the process to stop and the low order 8 bits will be set equal to 0177.

If the child process terminated due to an *exit* call, the low order 8 bits of status will be zero and the high order 8 bits will contain the low order 8 bits of the argument that the child process passed to *exit*; see *exit(2)*.

If the child process terminated due to a signal, the high order 8 bits of status will be zero and the low order 8 bits will contain the number of the signal that caused the termination. In addition, if the low order seventh bit (i.e., bit 200) is set, a "core image" will have been produced; see *signal(2)*.

If a parent process terminates without waiting for its child processes to terminate, the parent process ID of each child process is set to 1. This means the initialization process inherits the child processes; see *intro(2)*.

*Wait* will fail and return immediately if one or more of the following are true:

The calling process has no existing unwaited-for child processes. [ECHILD]

*Stat\_loc* points to an illegal address. [EFAULT]

**RETURN VALUE**

If *wait* returns due to the receipt of a signal, a value of  $-1$  is returned to the calling process and *errno* is set to EINTR. If *wait* returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of  $-1$  is returned and *errno* is set to indicate the error.

**SEE ALSO**

*exec(2)*, *exit(2)*, *fork(2)*, *pause(2)*, *signal(2)*.

**WARNING**

See *WARNING* in *signal(2)*.

## NAME

write — write on a file

## SYNOPSIS

```
int write (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

## DESCRIPTION

*Fildes* is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call.

*Write* attempts to write *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the *fildes*.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from *write*, the file pointer is incremented by the number of bytes actually written.

On devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is undefined.

If the *O\_APPEND* flag of the file status flags is set, the file pointer will be set to the end of the file prior to each write.

*Write* will fail and the file pointer will remain unchanged if one or more of the following are true:

*Fildes* is not a valid file descriptor open for writing. [EBADF]

An attempt is made to write to a pipe that is not open for reading by any process. [EPIPE and SIGPIPE signal]

An attempt was made to write a file that exceeds the process's file size limit or the maximum file size. See *ulimit*(2). [EFBIG]

*Buf* points outside the process's allocated address space. [EFAULT]

If a *write* requests that more bytes be written than there is room for (e.g., the *ulimit* (see *ulimit*(2)) or the physical end of a medium), only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes will give a failure return (except as noted below).

If the file being written is a pipe (or FIFO), no partial writes will be permitted. Thus, the write will fail if a write of *nbyte* bytes would exceed a limit.

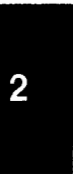
If the file being written is a pipe (or FIFO) and the *O\_NDELAY* flag of the file flag word is set, then write to a full pipe (or FIFO) will return a count of 0. Otherwise (*O\_NDELAY* clear), writes to a full pipe (or FIFO) will block until space becomes available.

## RETURN VALUE

Upon successful completion the number of bytes actually written is returned. Otherwise, *-1* is returned and *errno* is set to indicate the error.

## SEE ALSO

*creat*(2), *dup*(2), *lseek*(2), *open*(2), *pipe*(2), *ulimit*(2).





**NAME**

intro — introduction to subroutines and libraries

**SYNOPSIS****#include** <stdio.h>**#include** <math.h>**DESCRIPTION**

This section describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume. Certain major collections are identified by a letter after the section number:

- (3C) These functions, together with those of Section 2 and those marked (3S), constitute the Standard C Library *libc*, which is automatically loaded by the C compiler, *cc*(1). The link editor *ld*(1) searches this library under the *-lc* option. Declarations for some of these functions may be obtained from **#include** files indicated on the appropriate pages.
- (3F) These functions constitute the FORTRAN intrinsic function library, *libF77*. These functions are automatically available to the FORTRAN programmer and require no special invocation of the compiler.
- (3M) These functions constitute the Math Library, *libm*. They are automatically loaded as needed by the FORTRAN compiler *f77*(1). They are not automatically loaded by the C compiler, *cc*(1); however, the link editor searches this library under the *-lm* option. Declarations for these functions may be obtained from the **#include** file <math.h>.
- (3S) These functions constitute the "standard I/O package" (see *stdio*(3S)). These functions are in the library *libc*, already mentioned. Declarations for these functions may be obtained from the **#include** file <stdio.h>.
- (3X) Various specialized libraries. The files in which these libraries are found are given on the appropriate pages.

**DEFINITIONS**

A *character* is any bit pattern able to fit into a byte on the machine. The *null character* is a character with value 0, represented in the C language as '\0'. A *character array* is a sequence of characters. A *null-terminated character array* is a sequence of characters, the last of which is the *null character*. A *string* is a designation for a *null-terminated character array*. The *null string* is a character array containing only the null character. A *NULL pointer* is the value that is obtained by casting 0 into a pointer. The C language guarantees that this value will not match that of any legitimate pointer, so many functions that return pointers return it to indicate an error. *NULL* is defined as 0 in <stdio.h>; the user can include his own definition if he is not using <stdio.h>.

Many groups of FORTRAN intrinsic functions have *generic* function names that do not require explicit or implicit type declaration. The type of the function will be determined by the type of its argument(s). For example, the generic function *max* will return an integer value if given integer arguments (*max0*), a real value if given real arguments (*amax1*), or a double-precision value if given double-precision arguments (*dmax1*).

**FILES**

/lib/libc.a  
 /usr/lib/libF77.a  
 /lib/libm.a

**SEE ALSO**

ar(1), cc(1), f77(1), ld(1), nm(1), intro(2), stdio(3S).

**DIAGNOSTICS**

Functions in the Math Library (3M) may return the conventional values 0 or HUGE (the largest single-precision floating-point number) when the function is undefined for the given arguments or when the value is not representable. In these cases, the external variable *errno* (see *intro(2)*) is set to the value EDOM or ERANGE. As many of the FORTRAN intrinsic functions use the routines found in the Math Library, the same conventions apply.

**NAME**

*a64l*, *l64a* — convert between long integer and base-64 ASCII string

**SYNOPSIS**

```
long a64l (s)
char *s;
char *l64a (l)
long l;
```

**DESCRIPTION**

These functions are used to maintain numbers stored in *base-64* ASCII characters. This is a notation by which long integers can be represented by up to six characters; each character represents a “digit” in a radix-64 notation.

The characters used to represent “digits” are . for 0, / for 1, 0 through 9 for 2–11, A through Z for 12–37, and a through z for 38–63.

*A64l* takes a pointer to a null-terminated base-64 representation and returns a corresponding long value. If the string pointed to by *s* contains more than six characters, *a64l* will use the first six.

*L64a* takes a long argument and returns a pointer to the corresponding base-64 representation. If the argument is 0, *l64a* returns a pointer to a null string.

**BUGS**

The value returned by *l64a* is a pointer into a static buffer, the contents of which are overwritten by each call.

**NAME**

abort — generate an IOT fault

**SYNOPSIS**

int abort ( )

**DESCRIPTION**

*Abort* causes an IOT signal to be sent to the process. This usually results in termination with a core dump.

It is possible for *abort* to return control if SIGIOT is caught or ignored, in which case the value returned is that of the *kill(2)* system call.

**SEE ALSO**

adb(1), exit(2), kill(2), signal(2).

**DIAGNOSTICS**

If SIGIOT is neither caught nor ignored, and the current directory is writable, a core dump is produced and the message "abort — core dumped" is written by the shell.

**NAME**

abort — terminate Fortran program

**SYNOPSIS**

call abort ( )

**DESCRIPTION**

*Abort* terminates the program which calls it, closing all open files truncated to the current position of the file pointer.

**DIAGNOSTICS**

When invoked, *abort* prints "Fortran abort routine called" on the standard error output.

**SEE ALSO**

abort(3C).



**NAME**

`abs` — return integer absolute value

**SYNOPSIS**

```
int abs (i)
int i;
```

**DESCRIPTION**

*Abs* returns the absolute value of its integer operand.

**BUGS**

In two's-complement representation, the absolute value of the negative integer with largest magnitude is undefined. Some implementations trap this error, but others simply ignore it.

**SEE ALSO**

`floor(3M)`.

## NAME

`abs`, `iabs`, `dabs`, `cabs`, `zabs` — Fortran absolute value

## SYNOPSIS

```
integer i1, i2
real r1, r2
double precision dp1, dp2
complex cx1, cx2
double complex dx1, dx2

r2 = abs(r1)
i2 = iabs(i1)
i2 = abs(i1)

dp2 = dabs(dp1)
dp2 = abs(dp1)

cx2 = cabs(cx1)
cx2 = abs(cx1)

dx2 = zabs(dx1)
dx2 = abs(dx1)
```

## DESCRIPTION

*Abs* is the family of absolute value functions. *Iabs* returns the integer absolute value of its integer argument. *Dabs* returns the double-precision absolute value of its double-precision argument. *Cabs* returns the complex absolute value of its complex argument. *Zabs* returns the double-complex absolute value of its double-complex argument. The generic form *abs* returns the type of its argument.

## SEE ALSO

`floor(3M)`.

**NAME**

*acos*, *dacos* — Fortran arccosine intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
r2 = acos(r1)
dp2 = dacos(dp1)
dp2 = acos(dp1)
```

**DESCRIPTION**

*Acos* returns the real arccosine of its real argument. *Dacos* returns the double-precision arccosine of its double-precision argument. The generic form *acos* may be used with impunity as its argument will determine the type of the returned value.

**SEE ALSO**

*trig*(3M).

**NAME**

**aimag, dimag** — Fortran imaginary part of complex argument

**SYNOPSIS**

**real r**  
**complex cxr**  
**double precision dp**  
**double complex cxd**  
**r = aimag(cxr)**  
**dp = dimag(cxd)**

**DESCRIPTION**

*Aimag* returns the imaginary part of its single-precision complex argument.  
*Dimag* returns the double-precision imaginary part of its double-complex argument.

**NAME**

**aint, dint** — Fortran integer part intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
r2 = aint(r1)
dp2 = dint(dp1)
dp2 = aint(dp1)
```

**DESCRIPTION**

*Aint* returns the truncated value of its real argument in a real. *Dint* returns the truncated value of its double-precision argument as a double-precision value. *Aint* may be used as a generic function name, returning either a real or double-precision value depending on the type of its argument.

**NAME**

asin, dasin — Fortran arcsine intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
r2 = asin(r1)
dp2 = dasin(dp1)
dp2 = asin(dp1)
```

**DESCRIPTION**

*Asin* returns the real arcsine of its real argument. *Dasin* returns the double-precision arcsine of its double-precision argument. The generic form *asin* may be used with impunity as it derives its type from that of its argument.

**SEE ALSO**

trig(3M).

**NAME**

assert — verify program assertion

**SYNOPSIS**

```
#include <assert.h>
```

```
assert (expression)
```

```
int expression;
```

**DESCRIPTION**

This macro is useful for putting diagnostics into programs. When it is executed, if *expression* is false (zero), *assert* prints

```
“Assertion failed: expression, file xyz, line nnn”
```

on the standard error output and aborts. In the error message, *xyz* is the name of the source file and *nnn* the source line number of the *assert* statement.

Compiling with the preprocessor option `-DNDEBUG` (see *cpp*(1)), or with the preprocessor control statement `“#define NDEBUG”` ahead of the `“#include <assert.h>”` statement, will stop assertions from being compiled into the program.

**SEE ALSO**

*cpp*(1), *abort*(3C).

**NAME**

atan, datan — Fortran arctangent intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
r2 = atan(r1)
dp2 = datan(dp1)
dp2 = atan(dp1)
```

**DESCRIPTION**

*Atan* returns the real arctangent of its real argument. *Datan* returns the double-precision arctangent of its double-precision argument. The generic form *atan* may be used with a double-precision argument returning a double-precision value.

**SEE ALSO**

trig(3M).



**NAME**

atan2, datan2 — Fortran arctangent intrinsic function

**SYNOPSIS**

```
real r1, r2, r3
double precision dp1, dp2, dp3
r3 = atan2(r1, r2)
dp3 = datan2(dp1, dp2)
dp3 = atan2(dp1, dp2)
```

**DESCRIPTION**

*Atan2* returns the arctangent of *arg1/arg2* as a real value. *Datan2* returns the double-precision arctangent of its double-precision arguments. The generic form *atan2* may be used with impunity with double-precision arguments.

**SEE ALSO**

trig(3M).

**NAME**

`atof` — convert ASCII string to floating-point number

**SYNOPSIS**

```
double atof (nptr)
char *nptr;
```

**DESCRIPTION**

*Atof* converts a character string pointed to by *nptr* to a double-precision floating-point number. The first unrecognized character ends the conversion. *Atof* recognizes an optional string of white-space characters, then an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optionally signed integer. If the string begins with an unrecognized character, *atof* returns the value zero.

**DIAGNOSTICS**

When the correct value would overflow, *atof* returns HUGE, and sets *errno* to ERANGE. Zero is returned on underflow.

**SEE ALSO**

`scanf(3S)`.

## NAME

$j_0$ ,  $j_1$ ,  $j_n$ ,  $y_0$ ,  $y_1$ ,  $y_n$  — Bessel functions

## SYNOPSIS

```
#include <math.h>

double j0 (x)
double x;

double j1 (x)
double x;

double jn (n, x)
int n;
double x;

double y0 (x)
double x;

double y1 (x)
double x;

double yn (n, x)
int n;
double x;
```

## DESCRIPTION

$J_0$  and  $J_1$  return Bessel functions of  $x$  of the first kind of orders 0 and 1 respectively.  $J_n$  returns the Bessel function of  $x$  of the first kind of order  $n$ .

$Y_0$  and  $Y_1$  return the Bessel functions of  $x$  of the second kind of orders 0 and 1 respectively.  $Y_n$  returns the Bessel function of  $x$  of the second kind of order  $n$ . The value of  $x$  must be positive.

## DIAGNOSTICS

Non-positive arguments cause  $y_0$ ,  $y_1$  and  $y_n$  to return the value HUGE and to set *errno* to EDOM. They also cause a message indicating DOMAIN error to be printed on the standard error output; the process will continue.

These error-handling procedures may be changed with the function *matherr*(3M).

## SEE ALSO

*matherr*(3M).

**NAME**

and, or, xor, not, lshift, rshift — Fortran bitwise boolean functions

**SYNOPSIS**

integer i, j, k  
real a, b, c  
double precision dp1, dp2, dp3  
k = and(i, j)  
c = or(a, b)  
j = xor(i, a)  
j = not(i)  
k = lshift(i, j)  
k = rshift(i, j)

**DESCRIPTION**

The generic intrinsic boolean functions *and*, *or* and *xor* return the value of the binary operations on their arguments. *Not* is a unary operator returning the one's complement of its argument. *Lshift* and *rshift* return the value of the first argument shifted left or right, respectively, the number of times specified by the second (integer) argument.

The boolean functions are generic, that is, they are defined for all data types as arguments and return values. Where required, the compiler will generate appropriate type conversions.

**NOTE**

Although defined for all data types, use of boolean functions on any but integer data is bizarre and will probably result in unexpected consequences.

**BUGS**

The implementation of the shift functions may cause large shift values to deliver weird results.

**NAME**

bsearch — binary search

**SYNOPSIS**

```
char *bsearch ((char *) key, (char *) base, nel, sizeof (*key), compar)
unsigned nel;
int (*compar)( );
```

**DESCRIPTION**

*Bsearch* is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table indicating where a datum may be found. The table must be previously sorted in increasing order according to a provided comparison function. *Key* points to the datum to be sought in the table. *Base* points to the element at the base of the table. *Nel* is the number of elements in the table. *Compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero according as the first argument is to be considered less than, equal to, or greater than the second.

**DIAGNOSTICS**

A NULL pointer is returned if the key cannot be found in the table.

**NOTES**

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

**SEE ALSO**

lsearch(3C), hsearch(3C), qsort(3C), tsearch(3C).

**NAME**

`clock` — report CPU time used

**SYNOPSIS**

`long clock ( )`

**DESCRIPTION**

*Clock* returns the amount of CPU time (in microseconds) used since the first call to *clock*. The time reported is the sum of the user and system times of the calling process and its terminated child processes for which it has executed *wait(2)* or *system(3S)*.

The resolution of the clock is 10 milliseconds on Western Electric 3B processors, 16.667 milliseconds on Digital Equipment Corporation processors.

**SEE ALSO**

*times(2)*, *wait(2)*, *system(3S)*.

**BUGS**

The value returned by *clock* is defined in microseconds for compatibility with systems that have CPU clocks with much higher resolution. Because of this, the value returned will wrap around after accumulating only 2147 seconds of CPU time (about 36 minutes).

**NAME**

*conjg*, *dconjg* – Fortran complex conjugate intrinsic function

**SYNOPSIS**

**complex** *cx1*, *cx2*  
**double complex** *dx1*, *dx2*  
*cx2* = *conjg*(*cx1*)  
*dx2* = *dconjg*(*dx1*)

**DESCRIPTION**

*Conjg* returns the complex conjugate of its complex argument. *Dconjg* returns the double-complex conjugate of its double-complex argument.

**NAME**

*toupper*, *tolower*, *\_toupper*, *\_tolower*, *toascii* — translate characters

**SYNOPSIS**

```
#include <ctype.h>

int toupper (c)
int c;

int tolower (c)
int c;

int _toupper (c)
int c;

int _tolower (c)
int c;

int toascii (c)
int c;
```

**DESCRIPTION**

*Toupper* and *tolower* have as domain the range of *getc*(3S): the integers from -1 through 255. If the argument of *toupper* represents a lower-case letter, the result is the corresponding upper-case letter. If the argument of *tolower* represents an upper-case letter, the result is the corresponding lower-case letter. All other arguments in the domain are returned unchanged.

*\_toupper* and *\_tolower* are macros that accomplish the same thing as *toupper* and *tolower* but have restricted domains and are faster. *\_toupper* requires a lower-case letter as its argument; its result is the corresponding upper-case letter. *\_tolower* requires an upper-case letter as its argument; its result is the corresponding lower-case letter. Arguments outside the domain cause undefined results.

*Toascii* yields its argument with all bits turned off that are not part of a standard ASCII character; it is intended for compatibility with other systems.

**SEE ALSO**

*ctype*(3C), *getc*(3S).



**NAME**

*cos*, *dcos*, *ccos* — Fortran cosine intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2 = cos(r1)
dp2 = dcos(dp1)
dp2 = cos(dp1)
cx2 = ccos(cx1)
cx2 = cos(cx1)
```

**DESCRIPTION**

*Cos* returns the real cosine of its real argument. *Dcos* returns the double-precision cosine of its double-precision argument. *Ccos* returns the complex cosine of its complex argument. The generic form *cos* may be used with impunity as its returned type is determined by that of its argument.

**SEE ALSO**

trig(3M).

**NAME**

cosh, dcosh — Fortran hyperbolic cosine intrinsic function

**SYNOPSIS**

real r1, r2

double precision dp1, dp2

r2 = cosh(r1)

dp2 = dcosh(dp1)

dp2 = cosh(dp1)

**DESCRIPTION**

*Cosh* returns the real hyperbolic cosine of its real argument. *Dcosh* returns the double-precision hyperbolic cosine of its double-precision argument. The generic form *cosh* may be used to return the hyperbolic cosine in the type of its argument.

**SEE ALSO**

sinh(3M).

## NAME

`crypt`, `setkey`, `encrypt` — generate DES encryption

## SYNOPSIS

```
char *crypt (key, salt)
char *key, *salt;

void setkey (key)
char *key;

void encrypt (block, edflag)
char *block;
int edflag;
```

## DESCRIPTION

*Crypt* is the password encryption function. It is based on the NBS Data Encryption Standard (DES), with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

*Key* is a user's typed password. *Salt* is a two-character string chosen from the set [a-zA-Z0-9./]; this string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password. The first two characters are the salt itself.

The *setkey* and *encrypt* entries provide (rather primitive) access to the actual DES algorithm. The argument of *setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is set into the machine. This is the key that will be used with the above mentioned algorithm to encrypt or decrypt the string *block* with the function *encrypt*.

The argument to the *encrypt* entry is a character array of length 64 containing only the characters with numerical value 0 and 1. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by *setkey*. If *edflag* is zero, the argument is encrypted; if non-zero, it is decrypted.

## SEE ALSO

`login(1)`, `passwd(1)`, `getpass(3C)`, `passwd(4)`.

## BUGS

The return value points to static data that are overwritten by each call.

**NAME**

*ctermid* — generate file name for terminal

**SYNOPSIS**

```
#include <stdio.h>
char *ctermid(s)
char *s;
```

**DESCRIPTION**

*Ctermid* generates the path name of the controlling terminal for the current process, and stores it in a string.

If *s* is a NULL pointer, the string is stored in an internal static area, the contents of which are overwritten at the next call to *ctermid*, and the address of which is returned. Otherwise, *s* is assumed to point to a character array of at least `L_ctermid` elements; the path name is placed in this array and the value of *s* is returned. The constant `L_ctermid` is defined in the `<stdio.h>` header file.

**NOTES**

The difference between *ctermid* and *ttyname*(3C) is that *ttyname* must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while *ctermid* returns a string (`/dev/tty`) that will refer to the terminal if used as a file name. Thus *ttyname* is useful only if the process already has at least one file open to a terminal.

**SEE ALSO**

*ttyname*(3C).

## NAME

ctime, localtime, gmtime, asctime, tzset -- convert date and time to string

## SYNOPSIS

```
#include <time.h>
char *ctime (clock)
long *clock;
struct tm *localtime (clock)
long *clock;
struct tm *gmtime (clock)
long *clock;
char *asctime (tm)
struct tm *tm;
extern long timezone;
extern int daylight;
extern char *tzname[2];
void tzset ( )
```

## DESCRIPTION

*Ctime* converts a long integer, pointed to by *clock*, representing the time in seconds since 00:00:00 GMT, January 1, 1970, and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 01:03:52 1973\n\0
```

*Localtime* and *gmtime* return pointers to "tm" structures, described below. *Localtime* corrects for the time zone and possible Daylight Savings Time; *gmtime* converts directly to Greenwich Mean Time (GMT), which is the time the UNIX system uses.

*Asctime* converts a "tm" structure to a 26-character string, as shown in the above example, and returns a pointer to the string.

Declarations of all the functions and externals, and the "tm" structure, are in the *<time.h>* header file. The structure declaration is:

```
struct tm {
    int tm_sec; /* seconds (0 - 59) */
    int tm_min; /* minutes (0 - 59) */
    int tm_hour; /* hours (0 - 23) */
    int tm_mday; /* day of month (1 - 31) */
    int tm_mon; /* month of year (0 - 11) */
    int tm_year; /* year - 1900 */
    int tm_wday; /* day of week (Sunday = 0) */
    int tm_yday; /* day of year (0 - 365) */
    int tm_isdst;
};
```

*Tm\_isdst* is non-zero if Daylight Savings Time is in effect.

The external long variable *timezone* contains the difference, in seconds, between GMT and local standard time (in EST, *timezone* is 5\*60\*60); the external variable *daylight* is non-zero if and only if the standard U.S.A. Daylight Savings Time conversion should be applied. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended.

If an environment variable named TZ is present, *asctime* uses the contents of the variable to override the default time zone. The value of TZ must be a three-letter time zone name, followed by a number representing the difference between local time and Greenwich Mean Time in hours, followed by an optional three-letter name for a daylight time zone. For example, the setting for New Jersey would be EST5EDT. The effects of setting TZ are thus to change the values of the external variables *timezone* and *daylight*; in addition, the time zone names contained in the external variable

```
char *tzname[2] = { "EST", "EDT" };
```

are set from the environment variable TZ. The function *tzset* sets these external variables from TZ; *tzset* is called by *asctime* and may also be called explicitly by the user.

Note that in most installations, TZ is set by default when the user logs on, to a value in the local */etc/profile* file (see *profile(4)*).

**SEE ALSO**

*time(2)*, *getenv(3C)*, *profile(4)*, *environ(5)*.

**BUGS**

The return values point to static data whose content is overwritten by each call.

## NAME

*isalpha*, *isupper*, *islower*, *isdigit*, *isxdigit*, *isalnum*, *isspace*, *ispunct*, *isprint*, *isgraph*, *iscntrl*, *isascii* — classify characters

## SYNOPSIS

```
#include <ctype.h>
```

```
int isalpha (c)
```

```
int c;
```

```
...
```

## DESCRIPTION

These macros classify character-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value EOF (-1 — see *stdio*(3S)).

*isalpha*            *c* is a letter.

*isupper*           *c* is an upper-case letter.

*islower*           *c* is a lower-case letter.

*isdigit*            *c* is a digit [0-9].

*isxdigit*          *c* is a hexadecimal digit [0-9], [A-F] or [a-f].

*isalnum*            *c* is an alphanumeric (letter or digit).

*isspace*            *c* is a space, tab, carriage return, new-line, vertical tab, or form-feed.

*ispunct*            *c* is a punctuation character (neither control nor alphanumeric).

*isprint*            *c* is a printing character, code 040 (space) through 0176 (tilde).

*isgraph*           *c* is a printing character, like *isprint* except false for space.

*iscntrl*            *c* is a delete character (0177) or an ordinary control character (less than 040).

*isascii*            *c* is an ASCII character, code less than 0200.

## DIAGNOSTICS

If the argument to any of these macros is not in the domain of the function, the result is undefined.

## SEE ALSO

*ascii*(5).

**NAME**

`cuserid` — get character login name of the user

**SYNOPSIS**

```
#include <stdio.h>
```

```
char *cuserid (s)
```

```
char *s;
```

**DESCRIPTION**

*Cuserid* generates a character-string representation of the login name of the owner of the current process. If *s* is a NULL pointer, this representation is generated in an internal static area, the address of which is returned. Otherwise, *s* is assumed to point to an array of at least `L_cuserid` characters; the representation is left in this array. The constant `L_cuserid` is defined in the `<stdio.h>` header file.

**DIAGNOSTICS**

If the login name cannot be found, *cuserid* returns a NULL pointer; if *s* is not a NULL pointer, a null character (`\0`) will be placed at *s[0]*.

**SEE ALSO**

`getlogin(3C)`, `getpwent(3C)`.



## NAME

`dial` — establish an out-going terminal line connection

## SYNOPSIS

```
#include <dial.h>

int dial (call)
CALL *call;

void undial (fd)
int fd;
```

## DESCRIPTION

`Dial` returns a file-descriptor for a terminal line open for read/write. The argument to `dial` is a CALL structure (defined in the `<dial.h>` header file.

When finished with the terminal line, the calling program must invoke `undial` to release the semaphore that has been set during the allocation of the terminal device.

The CALL typedef in the `<dial.h>` header file is:

```
typedef struct {
    struct termio *attr;      /* pointer to termio attribute struct */
    int          baud;       /* transmission data rate */
    int          speed;      /* 212A modem: low=300, high=1200 */
    char         *line;      /* device name for out-going line */
    char         *telno;     /* pointer to tel-no digits string */
    int          modem;      /* specify modem control for direct lines */
} CALL;
```

The CALL element `speed` is intended only for use with an outgoing dialed call, in which case its value should be either 300 or 1200 to identify the 113A modem, or the high or low speed setting on the 212A modem. The CALL element `baud` is for the desired transmission baud rate. For example, one might set `baud` to 110 and `speed` to 300 (or 1200).

If the desired terminal line is a direct line, a string pointer to its device-name should be placed in the `line` element in the CALL structure. Legal values for such terminal device names are kept in the `L-devices` file. In this case, the value of the `baud` element need not be specified as it will be determined from the `L-devices` file.

The `telno` element is for a pointer to a character string representing the telephone number to be dialed. Such numbers may consist only of symbols described on the `acu(7)`. The termination symbol will be supplied by the `dial` function, and should not be included in the `telno` string passed to `dial` in the CALL structure.

The CALL element `modem` is used to specify modem control for direct lines. This element should be non-zero if modem control is required. The CALL element `attr` is a pointer to a `termio` structure, as defined in the `termio.h` header file. A NULL value for this pointer element may be passed to the `dial` function, but if such a structure is included, the elements specified in it will be set for the outgoing terminal line before the connection is established. This is often important for certain attributes such as parity and baud-rate.

## FILES

```
/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK.tty-device
```

## SEE ALSO

`uucp(1C)`, `alarm(2)`, `read(2)`, `write(2)`.

`acu(7)`, `termio(7)` in the *UNIX Administrator's Manual*.

## DIAGNOSTICS

On failure, a negative value indicating the reason for the failure will be returned. Mnemonics for these negative indices as listed here are defined in the `<dial.h>` header file.

<code>INTRPT</code>	<code>-1</code>	<code>/* interrupt occured */</code>
<code>D_HUNG</code>	<code>-2</code>	<code>/* dialer hung (no return from write) */</code>
<code>NO_ANS</code>	<code>-3</code>	<code>/* no answer within 10 seconds */</code>
<code>ILL_BD</code>	<code>-4</code>	<code>/* illegal baud-rate */</code>
<code>A_PROB</code>	<code>-5</code>	<code>/* acu problem (open() failure) */</code>
<code>L_PROB</code>	<code>-6</code>	<code>/* line problem (open() failure) */</code>
<code>NO_Ldv</code>	<code>-7</code>	<code>/* can't open LDEVS file */</code>
<code>DV_NT_A</code>	<code>-8</code>	<code>/* requested device not available */</code>
<code>DV_NT_K</code>	<code>-9</code>	<code>/* requested device not known */</code>
<code>NO_BD_A</code>	<code>-10</code>	<code>/* no device available at requested baud */</code>
<code>NO_BD_K</code>	<code>-11</code>	<code>/* no device known at requested baud */</code>

## WARNINGS

Including the `<dial.h>` header file automatically includes the `<termio.h>` header file.

The above routine uses `<stdio.h>`, which causes it to increase the size of programs, not otherwise using standard I/O, more than might be expected.

## BUGS

An `alarm(2)` system call for 3600 seconds is made (and caught) within the `dial` module for the purpose of "touching" the `LCK..` file and constitutes the device allocation semaphore for the terminal device. Otherwise, `uucp(1C)` may simply delete the `LCK..` entry on its 90-minute clean-up rounds. The alarm may go off while the user program is in a `read(2)` or `write(2)` system call, causing an apparent error return. If the user program expects to be around for an hour or more, error returns from `reads` should be checked for (`errno == EINTR`), and the `read` possibly reissued.

## NAME

*drand48*, *erand48*, *lrand48*, *nrand48*, *mrand48*, *jrand48*, *srand48*, *seed48*, *lcong48* — generate uniformly distributed pseudo-random numbers

## SYNOPSIS

```
double drand48 ( )
double erand48 (xsubi)
unsigned short xsubi[3];
long lrand48 ( )
long nrand48 (xsubi)
unsigned short xsubi[3];
long mrand48 ( )
long jrand48 (xsubi)
unsigned short xsubi[3];
void srand48 (seedval)
long seedval;
unsigned short *seed48 (seed16v)
unsigned short seed16v[3];
void lcong48 (param)
unsigned short param[7];
```

## DESCRIPTION

This family of functions generates pseudo-random numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.

Functions *drand48* and *erand48* return non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0).

Functions *lrand48* and *nrand48* return non-negative long integers uniformly distributed over the interval [0,  $2^{31}$ ).

Functions *mrand48* and *jrand48* return signed long integers uniformly distributed over the interval [ $-2^{31}$ ,  $2^{31}$ ).

Functions *srand48*, *seed48* and *lcong48* are initialization entry points, one of which should be invoked before either *drand48*, *lrand48* or *mrand48* is called. (Although it is not recommended practice, constant default initializer values will be supplied automatically if *drand48*, *lrand48* or *mrand48* is called without a prior call to an initialization entry point.) Functions *erand48*, *nrand48* and *jrand48* do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values,  $X_i$ , according to the linear congruential formula

$$X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0.$$

The parameter  $m = 2^{48}$ ; hence 48-bit integer arithmetic is performed. Unless *lcong48* has been invoked, the multiplier value  $a$  and the addend value  $c$  are given by

$$\begin{aligned} a &= 5DEECE66D_{16} = 273673163155_8 \\ c &= B_{16} = 13_8. \end{aligned}$$

The value returned by any of the functions *drand48*, *erand48*, *lrand48*, *nrand48*, *mrand48* or *jrand48* is computed by first generating the next 48-bit  $X_i$  in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of  $X_i$  and transformed into the returned value.

The functions *drand48*, *lrand48* and *mrnd48* store the last 48-bit  $X_i$  generated in an internal buffer; that is why they must be initialized prior to being invoked. The functions *erand48*, *nrnd48* and *jrnd48* require the calling program to provide storage for the successive  $X_i$  values in the array specified as an argument when the functions are invoked. That is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of  $X_i$  into the array and pass it as an argument. By using different arguments, functions *erand48*, *nrnd48* and *jrnd48* allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers, i.e., the sequence of numbers in each stream will *not* depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function *srand48* sets the high-order 32 bits of  $X_i$  to the 32 bits contained in its argument. The low-order 16 bits of  $X_i$  are set to the arbitrary value  $330E_{16}$ .

The initializer function *seed48* sets the value of  $X_i$  to the 48-bit value specified in the argument array. In addition, the previous value of  $X_i$  is copied into a 48-bit internal buffer, used only by *seed48*, and a pointer to this buffer is the value returned by *seed48*. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time — use the pointer to get at and store the last  $X_i$  value, and then use this value to reinitialize via *seed48* when the program is restarted.

The initialization function *lcng48* allows the user to specify the initial  $X_i$ , the multiplier value  $a$ , and the addend value  $c$ . Argument array elements *param*[0-2] specify  $X_i$ , *param*[3-5] specify the multiplier  $a$ , and *param*[6] specifies the 16-bit addend  $c$ . After *lcng48* has been called, a subsequent call to either *srand48* or *seed48* will restore the “standard” multiplier and addend values,  $a$  and  $c$ , specified on the previous page.

#### NOTES

The versions of these routines for the VAX-11 and PDP-11 are coded in assembly language for maximum speed. It requires approximately 80  $\mu$ sec on a VAX-11/780 and 130  $\mu$ sec on a PDP-11/70 to generate one pseudo-random number. On other computers, the routines are coded in portable C. The source code for the portable version can even be used on computers which do not have floating-point arithmetic. In such a situation, functions *drand48* and *erand48* do not exist; instead, they are replaced by the two new functions below.

```
long irand48 (m)
unsigned short m;
```

```
long krnd48 (xsubi, m)
unsigned short xsubi[3], m;
```

Functions *irand48* and *krnd48* return non-negative long integers uniformly distributed over the interval  $[0, m-1]$ .

#### SEE ALSO

rand(3C).

**NAME**

*ecvt*, *fcvt*, *gcvt* — convert floating-point number to string

**SYNOPSIS**

```
char *ecvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt (value, ndigit, buf)
double value;
char *buf;
```

**DESCRIPTION**

*Ecvt* converts *value* to a null-terminated string of *ndigit* digits and returns a pointer thereto. The low-order digit is rounded. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt* (negative means to the left of the returned digits). The decimal point is not included in the returned string. If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero.

*Fcvt* is identical to *ecvt*, except that the correct digit has been rounded for Fortran F-format output of the number of digits specified by *ndigit*.

*Gcvt* converts the *value* to a null-terminated string in the array pointed to by *buf* and returns *buf*. It attempts to produce *ndigit* significant digits in Fortran F-format if possible, otherwise E-format, ready for printing. A minus sign, if there is one, or a decimal point will be included as part of the returned string. Trailing zeros are suppressed.

**SEE ALSO**

`printf(3S)`.

**BUGS**

The return values point to static data whose content is overwritten by each call.

**NAME**

*end*, *etext*, *edata* — last locations in program

**SYNOPSIS**

```
extern end;
extern etext;
extern edata;
```

**DESCRIPTION**

These names refer neither to routines nor to locations with interesting contents. The address of *etext* is the first address above the program text, *edata* above the initialized data region, and *end* above the uninitialized data region.

When execution begins, the program break (the first location beyond the data) coincides with *end*, but the program break may be reset by the routines of *brk(2)*, *malloc(3C)*, standard input/output (*stdio(3S)*), the profile (*-p*) option of *cc(1)*, and so on. Thus, the current value of the program break should be determined by *sbrk(0)* (see *brk(2)*).

**SEE ALSO**

*brk(2)*, *malloc(3C)*.

**NAME**

erf, erfc — error function and complementary error function

**SYNOPSIS**

```
#include <math.h>
double erf (x)
double x;
double erfc (x)
double x;
```

**DESCRIPTION**

*Erf* returns the error function of  $x$ , defined as  $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ .

*Erfc*, which returns  $1.0 - erf(x)$ , is provided because of the extreme loss of relative accuracy if *erf(x)* is called for large  $x$  and the result subtracted from 1.0 (e.g. for  $x = 5$ , 12 places are lost).

**SEE ALSO**

exp(3M).

**NAME**

*exp*, *dexp*, *cexp* — Fortran exponential intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2 = exp(r1)

dp2 = dexp(dp1)
dp2 = exp(dp1)

cx2 = clog(cx1)
cx2 = exp(cx1)
```

**DESCRIPTION**

*Exp* returns the real exponential function  $e^x$  of its real argument. *Dexp* returns the double-precision exponential function of its double-precision argument. *Cexp* returns the complex exponential function of its complex argument. The generic function *exp* becomes a call to *dexp* or *cexp* as required, depending on the type of its argument.

**SEE ALSO**

*exp*(3M).



## NAME

`exp`, `log`, `log10`, `pow`, `sqrt` — exponential, logarithm, power, square root functions

## SYNOPSIS

```
#include <math.h>

double exp (x)
double x;

double log (x)
double x;

double log10 (x)
double x;

double pow (x, y)
double x, y;

double sqrt (x)
double x;
```

## DESCRIPTION

*Exp* returns  $e^x$ .

*Log* returns the natural logarithm of  $x$ . The value of  $x$  must be positive.

*Log10* returns the logarithm base ten of  $x$ . The value of  $x$  must be positive.

*Pow* returns  $x^y$ . The values of  $x$  and  $y$  may not both be zero. If  $x$  is non-positive,  $y$  must be an integer.

*Sqrt* returns the square root of  $x$ . The value of  $x$  may not be negative.

## DIAGNOSTICS

*Exp* returns HUGE when the correct value would overflow, and sets *errno* to ERANGE.

*Log* and *log10* return 0 and set *errno* to EDOM when  $x$  is non-positive. An error message is printed on the standard error output.

*Pow* returns 0 and sets *errno* to EDOM when  $x$  is non-positive and  $y$  is not an integer, or when  $x$  and  $y$  are both zero. In these cases a message indicating DOMAIN error is printed on the standard error output. When the correct value for *pow* would overflow, *pow* returns HUGE and sets *errno* to ERANGE.

*Sqrt* returns 0 and sets *errno* to EDOM when  $x$  is negative. A message indicating DOMAIN error is printed on the standard error output.

These error-handling procedures may be changed with the function *matherr*(3M).

## SEE ALSO

*hypot*(3M), *matherr*(3M), *sinh*(3M).

**NAME**

*fclose*, *fflush* — close or flush a stream

**SYNOPSIS**

```
#include <stdio.h>
```

```
int fclose (stream)
```

```
FILE *stream;
```

```
int fflush (stream)
```

```
FILE *stream;
```

**DESCRIPTION**

*Fclose* causes any buffered data for the named *stream* to be written out, and the *stream* to be closed.

*Fclose* is performed automatically for all open files upon calling *exit*(2).

*Fflush* causes any buffered data for the named *stream* to be written to that file. The *stream* remains open.

**DIAGNOSTICS**

These functions return 0 for success, and EOF if any error (such as trying to write to a file that has not been opened for writing) was detected.

**SEE ALSO**

*close*(2), *exit*(2), *fopen*(3S), *setbuf*(3S).

## NAME

`ferror`, `feof`, `clearerr`, `fileno` — stream status inquiries

## SYNOPSIS

```
#include <stdio.h>

int feof (stream)
FILE
*stream;

int ferror (stream)
FILE
*stream;

void clearerr (stream)
FILE
*stream;

int fileno(stream)
FILE
*stream;
```

## DESCRIPTION

*Feof* returns non-zero when EOF has previously been detected reading the named input *stream*, otherwise zero.

*Ferror* returns non-zero when an I/O error has previously occurred reading from or writing to the named *stream*, otherwise zero.

*Clearerr* resets the error indicator and EOF indicator to zero on the named *stream*.

*Fileno* returns the integer file descriptor associated with the named *stream*; see *open(2)*.

## NOTE

All these functions are implemented as macros; they cannot be declared or redeclared.

## SEE ALSO

*open(2)*, *fopen(3S)*.

**NAME**

floor, ceil, fmod, fabs — floor, ceiling, remainder, absolute value functions

**SYNOPSIS**

```
#include <math.h>
```

```
double floor (x)
```

```
double x;
```

```
double ceil (x)
```

```
double x;
```

```
double fmod (x, y)
```

```
double x, y;
```

```
double fabs (x)
```

```
double x;
```

**DESCRIPTION**

*Floor* returns the largest integer (as a double-precision number) not greater than  $x$ .

*Ceil* returns the smallest integer not less than  $x$ .

*Fmod* returns  $x$  if  $y$  is zero, otherwise the number  $f$  with the same sign as  $x$ , such that  $x = iy + f$  for some integer  $i$ , and  $|f| < |y|$ .

*Fabs* returns  $|x|$ .

**SEE ALSO**

abs(3C).

## NAME

fopen, freopen, fdopen — open a stream

## SYNOPSIS

```
#include <stdio.h>

FILE *fopen (file-name, type)
char *file-name, *type;

FILE *freopen (file-name, type, stream)
char *file-name, *type;
FILE *stream;

FILE *fdopen (fdes, type)
int fdes;
char *type;
```

## DESCRIPTION

*Fopen* opens the file named by *file-name* and associates a *stream* with it. *Fopen* returns a pointer to the FILE structure associated with the *stream*.

*File-name* points to a character string that contains the name of the file to be opened.

*Type* is a character string having one of the following values:

"r"	open for reading
"w"	truncate or create for writing
"a"	append; open for writing at end of file, or create for writing
"r+"	open for update (reading and writing)
"w+"	truncate or create for update
"a+"	append; open or create for update at end-of-file

*Freopen* substitutes the named file in place of the open *stream*. The original *stream* is closed, regardless of whether the open ultimately succeeds. *Freopen* returns a pointer to the FILE structure associated with *stream*.

*Freopen* is typically used to attach the preopened *streams* associated with *stdin*, *stdout* and *stderr* to other files.

*Fdopen* associates a *stream* with a file descriptor obtained from *open*, *dup*, *creat*, or *pipe(2)*, which will open files but not return pointers to a FILE structure *stream* which are necessary input for many of the section 3S library routines. The *type* of *stream* must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting *stream*. However, output may not be directly followed by input without an intervening *fseek* or *rewind*, and input may not be directly followed by output without an intervening *fseek*, *rewind*, or an input operation which encounters end-of-file.

When a file is opened for append (i.e., when *type* is "a" or "a+"), it is impossible to overwrite information already in the file. *Fseek* may be used to reposition the file pointer to any position in the file, but when output is written to the file the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output. If two separate processes open the same file for append, each process may write freely to the file without fear of destroying output being written by the other. The output from the two processes will be intermixed in the file in the order in which it is written.

**SEE ALSO**

open(2), fclose(3S).

**DIAGNOSTICS**

*Fopen* and *freopen* return a NULL pointer on failure.

## NAME

fread, fwrite — binary input/output

## SYNOPSIS

```
#include <stdio.h>

int fread (ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;

int fwrite (ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;
```

## DESCRIPTION

*Fread* copies, into an array beginning at *ptr*, *nitems* items of data from the named input *stream*, where an item of data is a sequence of bytes (not necessarily terminated by a null byte) of length *size*. *Fread* stops appending bytes if an end-of-file or error condition is encountered while reading *stream*, or if *nitems* items have been read. *Fread* leaves the file pointer in *stream*, if defined, pointing to the byte following the last byte read if there is one. *Fread* does not change the contents of *stream*.

*Fwrite* appends at most *nitems* items of data from the the array pointed to by *ptr* to the named output *stream*. *Fwrite* stops appending when it has appended *nitems* items of data or if an error condition is encountered on *stream*. *Fwrite* does not change the contents of the array pointed to by *ptr*.

The variable *size* is typically *sizeof(\*ptr)* where the pseudo-function *sizeof* specifies the length of an item pointed to by *ptr*. If *ptr* points to a data type other than *char* it should be cast into a pointer to *char*.

## SEE ALSO

read(2), write(2), fopen(3S),getc(3S), gets(3S), printf(3S), putc(3S), puts(3S), scanf(3S).

## DIAGNOSTICS

*Fread* and *fwrite* return the number of items read or written. If *nitems* is non-positive, no characters are read or written and 0 is returned by both *fread* and *fwrite*.

**NAME**

*frexp*, *ldexp*, *modf* — manipulate parts of floating-point numbers

**SYNOPSIS**

```
double frexp (value, eptr)
double value;
int *eptr;

double ldexp (value, exp)
double value;
int exp;

double modf (value, iptr)
double value, *iptr;
```

**DESCRIPTION**

Every non-zero number can be written uniquely as  $x * 2^n$ , where the “mantissa” (fraction)  $x$  is in the range  $0.5 \leq |x| < 1.0$ , and the “exponent”  $n$  is an integer. *Frexp* returns the mantissa of a double *value*, and stores the exponent indirectly in the location pointed to by *eptr*.

*Ldexp* returns the quantity  $value * 2^{exp}$ .

*Modf* returns the signed fractional part of *value* and stores the integral part indirectly in the location pointed to by *iptr*.

**DIAGNOSTICS**

If *ldexp* would cause overflow, **HUGE** is returned and *errno* is set to **ERANGE**.



**NAME**

`fseek`, `rewind`, `ftell` — reposition a file pointer in a stream

**SYNOPSIS**

```
#include <stdio.h>

int fseek (stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

void rewind (stream)
FILE *stream;

long ftell (stream)
FILE *stream;
```

**DESCRIPTION**

*Fseek* sets the position of the next input or output operation on the *stream*. The new position is at the signed distance *offset* bytes from the beginning, from the current position, or from the end of the file, according as *ptrname* has the value 0, 1, or 2.

*Rewind(stream)* is equivalent to *fseek(stream, 0L, 0)*, except that no value is returned.

*Fseek* and *rewind* undo any effects of *ungetc*(3S).

After *fseek* or *rewind*, the next operation on a file opened for update may be either input or output.

*Ftell* returns the offset of the current byte relative to the beginning of the file associated with the named *stream*.

**SEE ALSO**

`lseek`(2), `fopen`(3S).

**DIAGNOSTICS**

*Fseek* returns non-zero for improper seeks, otherwise zero. An improper seek can be, for example, an *fseek* done on a file that has not been opened via *fopen*; in particular, *fseek* may not be used on a terminal, or on a file opened via *popen*(3S).

**WARNING**

Although on UNIX an offset returned by *ftell* is measured in bytes, and it is permissible to seek to positions relative to that offset, portability to non-UNIX systems requires that an offset be used by *fseek* directly. Arithmetic may not meaningfully be performed on such a offset, which is not necessarily measured in bytes.

**NAME**

*ftw* — walk a file tree

**SYNOPSIS**

```
#include <ftw.h>

int ftw (path, fn, depth)
char *path;
int (*fn) ( );
int depth;
```

**DESCRIPTION**

*Ftw* recursively descends the directory hierarchy rooted in *path*. For each object in the hierarchy, *ftw* calls *fn*, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a *stat* structure (see *stat(2)*) containing information about the object, and an integer. Possible values of the integer, defined in the <ftw.h> header file, are FTW\_F for a file, FTW\_D for a directory, FTW\_DNR for a directory that cannot be read, and FTW\_NS for an object for which *stat* could not successfully be executed. If the integer is FTW\_DNR, descendants of that directory will not be processed. If the integer is FTW\_NS, the *stat* structure will contain garbage. An example of an object that would cause FTW\_NS to be passed to *fn* would be a file in a directory with read but without execute (search) permission.

*Ftw* visits a directory before visiting any of its descendants.

The tree traversal continues until the tree is exhausted, an invocation of *fn* returns a nonzero value, or some error is detected within *ftw* (such as an I/O error). If the tree is exhausted, *ftw* returns zero. If *fn* returns a nonzero value, *ftw* stops its tree traversal and returns whatever value was returned by *fn*. If *ftw* detects an error, it returns -1, and sets the error type in *errno*.

*Ftw* uses one file descriptor for each level in the tree. The *depth* argument limits the number of file descriptors so used. If *depth* is zero or negative, the effect is the same as if it were 1. *Depth* must not be greater than the number of file descriptors currently available for use. *Ftw* will run more quickly if *depth* is at least as large as the number of levels in the tree.

**SEE ALSO**

*stat(2)*, *malloc(3C)*.

**BUGS**

Because *ftw* is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

It could be made to run faster and use less storage on deep structures at the cost of considerable complexity.

*Ftw* uses *malloc(3C)* to allocate dynamic storage during its operation. If *ftw* is forcibly terminated, such as by *longjmp* being executed by *fn* or an interrupt routine, *ftw* will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a nonzero value at its next invocation.

## NAME

int, ifix, idint, real, float, sngl, dble, cmplx, dcplx, ichar, char — explicit Fortran type conversion

## SYNOPSIS

```

integer i, j
real r, s
double precision dp, dq
complex cx
double complex dcx
character*! ch

i = int(r)
i = int(dp)
i = int(cx)
i = int(dcx)
i = ifix(r)
i = idint(dp)

r = real(i)
r = real(dp)
r = real(cx)
r = real(dcx)
r = float(i)
r = sngl(dp)

dp = dble(i)
dp = dble(r)
dp = dble(cx)
dp = dble(dcx)

cx = cmplx(i)
cx = cmplx(i, j)
cx = cmplx(r)
cx = cmplx(r, s)
cx = cmplx(dp)
cx = cmplx(dp, dq)
cx = cmplx(dcx)

dcx = dcplx(i)
dcx = dcplx(i, j)
dcx = dcplx(r)
dcx = dcplx(r, s)
dcx = dcplx(dp)
dcx = dcplx(dp, dq)
dcx = dcplx(cx)

i = ichar(ch)
ch = char(i)

```

## DESCRIPTION

These functions perform conversion from one data type to another.

**int** converts to *integer* form its *real*, *double precision*, *complex*, or *double complex* argument. If the argument is *real* or *double precision*, **int** returns the integer whose magnitude is the largest integer that does not exceed the magnitude of the argument and whose sign is the same as the sign of the argument (i.e. truncation). For complex types, the above rule is applied to the real part. **ifix** and **idint** convert only *real* and *double precision* arguments respectively.

**real** converts to *real* form an *integer*, *double precision*, *complex*, or *double complex* argument. If the argument is *double precision* or *double complex*, as much precision is kept as is possible. If the argument is one of the complex types, the real part is returned. **float** and **sngl** convert only *integer* and *double precision* arguments respectively.

**dble** converts any *integer*, *real*, *complex*, or *double complex* argument to *double precision* form. If the argument is of a complex type, the real part is returned.

**cmplx** converts its *integer*, *real*, *double precision*, or *double complex* argument(s) to *complex* form.

**dcmplx** converts to *double complex* form its *integer*, *real*, *double precision*, or *complex* argument(s).

Either one or two arguments may be supplied to **cmplx** and **dcmplx**. If there is only one argument, it is taken as the real part of the complex type and a imaginary part of zero is supplied. If two arguments are supplied, the first is taken as the real part and the second as the imaginary part.

**ichar** converts from a character to an integer depending on the character's position in the collating sequence.

**char** returns the character in the *i*th position in the processor collating sequence where *i* is the supplied argument.

For a processor capable of representing *n* characters,

**ichar(char(i)) = i** for  $0 \leq i < n$ , and

**char(ichar(ch)) = ch** for any representable character *ch*.

**NAME**

gamma — log gamma function

**SYNOPSIS**

```
#include <math.h>
extern int signgam;
double gamma (x)
double x;
```

**DESCRIPTION**

*Gamma* returns  $\ln(|\Gamma(x)|)$ , where  $\Gamma(x)$  is defined as  $\int_0^{\infty} e^{-t} t^{x-1} dt$ . The sign of  $\Gamma(x)$  is returned in the external integer *signgam*. The argument *x* may not be a non-positive integer.

The following C program fragment might be used to calculate  $\Gamma$ :

```
if ((y = gamma(x)) > LOGHUGE)
    error();
y = signgam * exp(y);
```

where LOGHUGE is the least value that causes *exp(3M)* to return a range error.

**DIAGNOSTICS**

For non-negative integer arguments HUGE is returned, and *errno* is set to EDOM. A message indicating DOMAIN error is printed on the standard error output.

If the correct value would overflow, *gamma* returns HUGE and sets *errno* to ERANGE.

These error-handling procedures may be changed with the function *matherr(3M)*.

**SEE ALSO**

*exp(3M)*, *matherr(3M)*.

**NAME**

*getarg* — return Fortran command-line argument

**SYNOPSIS**

**character\****N* *c*

**integer** *i*

*getarg*(*i*, *c*)

**DESCRIPTION**

*Getarg* returns the *i*-th command-line argument of the current process. Thus, if a program were invoked via

foo arg1 arg2 arg3

*getarg*(2, *c*) would return the string "arg2" in the character variable *c*.

**SEE ALSO**

*getopt*(3C).

**NAME**

*getc*, *getchar*, *fgetc*, *getw* — get character or word from stream

**SYNOPSIS**

```
#include <stdio.h>

int getc (stream)
FILE *stream;

int getchar ( )

int fgetc (stream)
FILE *stream;

int getw (stream)
FILE *stream;
```

**DESCRIPTION**

*Getc* returns the next character (i.e. byte) from the named input *stream*. It also moves the file pointer, if defined, ahead one character in *stream*. *Getc* is a macro and so cannot be used if a function is necessary; for example one cannot have a function pointer point to it.

*Getchar* returns the next character from the standard input stream, *stdin*. As in the case of *getc*, *getchar* is a macro.

*Fgetc* performs the same function as *getc*, but is a genuine function. *Fgetc* runs more slowly than *getc*, but takes less space per invocation.

*Getw* returns the next word (i.e. integer) from the named input *stream*. The size of a word varies from machine to machine. It returns the constant EOF upon end-of-file or error, but as that is a valid integer value, *feof* and *ferror*(3S) should be used to check the success of *getw*. *Getw* increments the associated file pointer, if defined, to point to the next word. *Getw* assumes no special alignment in the file.

**SEE ALSO**

*fclose*(3S), *ferror*(3S), *fopen*(3S), *fread*(3S), *gets*(3S), *putc*(3S), *scanf*(3S).

**DIAGNOSTICS**

These functions return the integer constant EOF at end-of-file or upon an error.

**BUGS**

Because it is implemented as a macro, *getc* treats incorrectly a *stream* argument with side effects. In particular, *getc*(\*f++) doesn't work sensibly. *Fgetc* should be used instead.

Because of possible differences in word length and byte ordering, files written using *putw* are machine-dependent, and may not be read using *getw* on a different processor.

**NAME**

*getcwd* — get path-name of current working directory

**SYNOPSIS**

```
char *getcwd (buf, size)
char *buf;
int size;
```

**DESCRIPTION**

*Getcwd* returns a pointer to the current directory path-name. The value of *size* must be at least two greater than the length of the path-name to be returned.

If *buf* is a NULL pointer, *getcwd* will obtain *size* bytes of space using *malloc*(3C). In this case, the pointer returned by *getcwd* may be used as the argument in a subsequent call to *free*.

The function is implemented by using *popen*(3S) to pipe the output of the *pwd*(1) command into the specified string space.

**EXAMPLE**

```
char *cwd, *getcwd();
.
.
.
if ((cwd = getcwd((char *)NULL, 64)) == NULL) {
    perror("pwd");
    exit(1);
}
printf("%s\n", cwd);
```

**SEE ALSO**

*pwd*(1), *malloc*(3C), *popen*(3S).

**DIAGNOSTICS**

Returns NULL with *errno* set if *size* is not large enough, or if an error occurs in a lower-level function.



**NAME**

`getenv` — return value for environment name

**SYNOPSIS**

```
char *getenv (name)
char *name;
```

**DESCRIPTION**

*Getenv* searches the environment list (see *environ(5)*) for a string of the form *name=value*, and returns a pointer to the *value* in the current environment if such a string is present, otherwise a NULL pointer.

**SEE ALSO**

*environ(5)*.

**NAME**

`getenv` — return Fortran environment variable

**SYNOPSIS**

**character\*N c**

`getenv("TMPDIR", c)`

**DESCRIPTION**

*Getenv* returns the character-string value of the environment variable represented by its first argument into the character variable of its second argument. If no such environment variable exists, all blanks will be returned.

**SEE ALSO**

`getenv(3C)`, `environ(5)`.

## NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent — get group file entry

## SYNOPSIS

```
#include <grp.h>
struct group *getgrent ( )
struct group *getgrgid (gid)
int gid;
struct group *getgrnam (name)
char *name;
void setgrent ( )
void endgrent ( )
```

## DESCRIPTION

*Getgrent*, *getgrgid* and *getgrnam* each return pointers to an object with the following structure containing the broken-out fields of a line in the */etc/group* file. Each line contains a “group” structure, defined in the *<grp.h>* header file.

```
struct group {
    char   *gr_name; /* the name of the group */
    char   *gr_passwd; /* the encrypted group password */
    int    gr_gid; /* the numerical group ID */
    char   **gr_mem; /* vector of pointers to member names */
};
```

*Getgrent* when first called returns a pointer to the first group structure in the file; thereafter, it returns a pointer to the next group structure in the file; so, successive calls may be used to search the entire file. *Getgrgid* searches from the beginning of the file until a numerical group id matching *gid* is found and returns a pointer to the particular structure in which it was found. *Getgrnam* searches from the beginning of the file until a group name matching *name* is found and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to *setgrent* has the effect of rewinding the group file to allow repeated searches. *Endgrent* may be called to close the group file when processing is complete.

## FILES

*/etc/group*

## SEE ALSO

*getlogin(3C)*, *getpwent(3C)*, *group(4)*.

## DIAGNOSTICS

A NULL pointer is returned on EOF or error.

## WARNING

The above routines use *<stdio.h>*, which causes them to increase the size of programs, not otherwise using standard I/O, more than might be expected.

## BUGS

All information is contained in a static area, so it must be copied if it is to be saved.

**NAME**

getlogin — get login name

**SYNOPSIS**

```
char *getlogin ( );
```

**DESCRIPTION**

*Getlogin* returns a pointer to the login name as found in */etc/utmp*. It may be used in conjunction with *getpwnam* to locate the correct password file entry when the same user ID is shared by several login names.

If *getlogin* is called within a process that is not attached to a terminal, it returns a NULL pointer. The correct procedure for determining the login name is to call *cuserid*, or to call *getlogin* and if it fails to call *getpwuid*.

**FILES**

*/etc/utmp*

**SEE ALSO**

*cuserid(3S)*, *getgrent(3C)*, *getpwent(3C)*, *utmp(4)*.

**DIAGNOSTICS**

Returns the NULL pointer if *name* not found.

**BUGS**

The return values point to static data whose content is overwritten by each call.

## NAME

getopt — get option letter from argument vector

## SYNOPSIS

```
int getopt (argc, argv, optstring)
int argc;
char **argv;
char *optstring;

extern char *optarg;
extern int optind;
```

## DESCRIPTION

*Getopt* returns the next option letter in *argv* that matches a letter in *optstring*. *Optstring* is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument that may or may not be separated from it by white space. *Optarg* is set to point to the start of the option argument on return from *getopt*.

*Getopt* places in *optind* the *argv* index of the next argument to be processed. Because *optind* is external, it is normally initialized to zero automatically before the first call to *getopt*.

When all options have been processed (i.e., up to the first non-option argument), *getopt* returns EOF. The special option -- may be used to delimit the end of the options; EOF will be returned, and -- will be skipped.

## DIAGNOSTICS

*Getopt* prints an error message on *stderr* and returns a question mark (?) when it encounters an option letter not included in *optstring*.

## WARNING

The above routine uses `<stdio.h>`, which causes it to increase the size of programs, not otherwise using standard I/O, more than might be expected.

## EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options a and b, and the options f and o, both of which require arguments:

```
main (argc, argv)
int argc;
char **argv;
{
    int c;
    extern int optind;
    extern char *optarg;
    :
    while ((c = getopt (argc, argv, "abf:o:")) != EOF)
        switch (c) {
            case 'a':
                if (bflg)
                    errflg++;
                else
                    aflg++;
                break;
            case 'b':
                if (aflg)
                    errflg++;
                else
                    bproc( );
```

```
        break;
    case 'f':
        ifile = optarg;
        break;
    case 'o':
        ofile = optarg;
        bufsiza = 512;
        break;
    case '?':
        errflg++;
    }
    if (errflg) {
        fprintf(stderr, "usage: . . . ");
        exit (2);
    }
    for ( ; optind < argc; optind++) {
        if (access (argv[optind], 4)) {
            :
        }
    }
```

**SEE ALSO**  
getopt(1).

**NAME**

getpass — read a password

**SYNOPSIS**

```
char *getpass (prompt)
char *prompt;
```

**DESCRIPTION**

*Getpass* reads up to a newline or EOF from the file */dev/tty*, after prompting on the standard error output with the null-terminated string *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters. If */dev/tty* cannot be opened, a NULL pointer is returned. An interrupt will terminate input and send an interrupt signal to the calling program before returning.

**FILES**

*/dev/tty*

**SEE ALSO**

crypt(3C).

**WARNING**

The above routine uses `<stdio.h>`, which causes it to increase the size of programs, not otherwise using standard I/O, more than might be expected.

**BUGS**

The return value points to static data whose content is overwritten by each call.

**NAME**

`getpw` — get name from UID

**SYNOPSIS**

```
int getpw (uid, buf)
int uid;
char *buf;
```

**DESCRIPTION**

*Getpw* searches the password file for a user *uid* number that equals *uid*, copies the line of the password file in which *uid* was found into the array pointed to by *buf*, and returns 0. *Getpw* returns non-zero if *uid* cannot be found.

This routine is included only for compatibility with prior systems and should not be used; see *getpwent*(3C) for routines to use instead.

**FILES**

`/etc/passwd`

**SEE ALSO**

*getpwent*(3C), *passwd*(4).

**DIAGNOSTICS**

*Getpw* returns non-zero on error.

**WARNING**

The above routine uses `<stdio.h>`, which causes it to increase the size of programs, not otherwise using standard I/O, more than might be expected.



## NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent — get password file entry

## SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent ( )
struct passwd *getpwuid (uid)
int uid;
struct passwd *getpwnam (name)
char *name;
void setpwent ( )
void endpwent ( )
```

## DESCRIPTION

*Getpwent*, *getpwuid* and *getpwnam* each returns a pointer to an object with the following structure containing the broken-out fields of a line in the */etc/passwd* file. Each line in the file contains a “passwd” structure, declared in the *<pwd.h>* header file:

```
struct passwd {
    char    *pw_name;
    char    *pw_passwd;
    int     pw_uid;
    int     pw_gid;
    char    *pw_age;
    char    *pw_comment;
    char    *pw_gecos;
    char    *pw_dir;
    char    *pw_shell;
};

struct comment {
    char    *c_dept;
    char    *c_name;
    char    *c_acct;
    char    *c_bin;
};
```

This structure is declared in *<pwd.h>* so it is not necessary to redeclare it.

The *pw\_comment* field is unused; the others have meanings described in *passwd(4)*.

*Getpwent* when first called returns a pointer to the first passwd structure in the file; thereafter, it returns a pointer to the next passwd structure in the file; so successive calls can be used to search the entire file. *Getpwuid* searches from the beginning of the file until a numerical user id matching *uid* is found and returns a pointer to the particular structure in which it was found. *Getpwnam* searches from the beginning of the file until a login name matching *name* is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to *setpwent* has the effect of rewinding the password file to allow repeated searches. *Endpwent* may be called to close the password file when processing is complete.

**FILES**

/etc/passwd

**SEE ALSO**

getlogin(3C), getgrent(3C), passwd(4).

**DIAGNOSTICS**

A NULL pointer is returned on EOF or error.

**WARNING**

The above routines use `<stdio.h>`, which causes them to increase the size of programs, not otherwise using standard I/O, more than might be expected.

**BUGS**

All information is contained in a static area, so it must be copied if it is to be saved.

**NAME**

gets, fgets — get a string from a stream

**SYNOPSIS**

```
#include <stdio.h>
char *gets (s)
char *s;
char *fgets (s, n, stream)
char *s;
int n;
FILE *stream;
```

**DESCRIPTION**

*Gets* reads characters from the standard input stream, *stdin*, into the array pointed to by *s*, until a new-line character is read or an end-of-file condition is encountered. The new-line character is discarded and the string is terminated with a null character.

*Fgets* reads characters from the *stream* into the array pointed to by *s*, until *n*−1 characters are read, or a new-line character is read and transferred to *s*, or an end-of-file condition is encountered. The string is then terminated with a null character.

**SEE ALSO**

error(3S), fopen(3S), fread(3S), getc(3S), scanf(3S).

**DIAGNOSTICS**

If end-of-file is encountered and no characters have been read, no characters are transferred to *s* and a NULL pointer is returned. If a read error occurs, such as trying to use these functions on a file that has not been opened for reading, a NULL pointer is returned. Otherwise *s* is returned.

## NAME

gettent, getutid, getutline, pututline, setutent, endutent, utmpname — access utmp file entry

## SYNOPSIS

```
#include <utmp.h>

struct utmp *gettent ( )
struct utmp *getutid (id)
struct utmp *id;
struct utmp *getutline (line)
struct utmp *line;
void pututline (utmp)
struct utmp *utmp;
void setutent ( )
void endutent ( )
void utmpname (file)
char *file;
```

## DESCRIPTION

*Gettent*, *getutid* and *getutline* each return a pointer to a structure of the following type:

```
struct utmp {
    char    ut_user[8];           /* User login name */
    char    ut_id[4];            /* /etc/inittab id (usually line #) */
    char    ut_line[12];         /* device name (console, lxxx) */
    short   ut_pid;              /* process id */
    short   ut_type;             /* type of entry */
    struct  exit_status {
        short e_termination; /* Process termination status */
        short e_exit;         /* Process exit status */
    } ut_exit;                  /* The exit status of a process
                                * marked as DEAD_PROCESS. */
    time_t  ut_time;            /* time entry was made */
};
```

*Gettent* reads in the next entry from a *utmp*-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.

*Getutid* searches forward from the current point in the *utmp* file until it finds an entry with a *ut\_type* matching *id*—>*ut\_type* if the type specified is RUN\_LVL, BOOT\_TIME, OLD\_TIME or NEW\_TIME. If the type specified in *id* is INIT\_PROCESS, LOGIN\_PROCESS, USER\_PROCESS or DEAD\_PROCESS, then *getutid* will return a pointer to the first entry whose type is one of these four and whose *ut\_id* field matches *id*—>*ut\_id*. If the end of file is reached without a match, it fails.

*Getutline* searches forward from the current point in the *utmp* file until it finds an entry of the type LOGIN\_PROCESS or USER\_PROCESS which also has a *ut\_line* string matching the *line*—>*ut\_line* string. If the end of file is reached without a match, it fails.

*Pututline* writes out the supplied *utmp* structure into the *utmp* file. It uses *getutid* to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of *pututline* will have searched for the proper entry using one of the *getut* routines. If so, *pututline* will not search. If *pututline* does not find a matching slot for the

new entry, it will add a new entry to the end of the file.

*Setutent* resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.

*Endutent* closes the currently open file.

*Utmpname* allows the user to change the name of the file examined, from */etc/utmp* to any other file. It is most often expected that this other file will be */etc/wtmp*. If the file doesn't exist, this will not be apparent until the first attempt to reference the file is made. *Utmpname* does not open the file. It just closes the old file if it is currently open and saves the new file name.

#### FILES

*/etc/utmp*

*/etc/wtmp*

#### SEE ALSO

*ttyslot(3C)*, *utmp(4)*.

#### DIAGNOSTICS

A NULL pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

#### COMMENTS

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. Each call to either *getutid* or *getutline* sees the routine examine the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason to use *getutline* to search for multiple occurrences, it would be necessary to zero out the static after each success, or *getutline* would just return the same pointer over and over again. There is one exception to the rule about removing the structure before further reads are done. The implicit read done by *pututline* if it finds that it isn't already at the correct place in the file will not hurt the contents of the static structure returned by the *getutent*, *getutid* or *getutline* routines, if the user has just modified those contents and passed the pointer back to *pututline*.

These routines use buffered standard I/O for input, but *pututline* uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the *utmp* and *wtmp* files.

## NAME

`hsearch`, `hcreate`, `hdestroy` — manage hash search tables

## SYNOPSIS

```
#include <search.h>
ENTRY *hsearch (item, action)
ENTRY item;
ACTION action;

int hcreate (nel)
unsigned nel;

void hdestroy ( )
```

## DESCRIPTION

*Hsearch* is a hash-table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found. *Item* is a structure of type `ENTRY` (defined in the `<search.h>` header file) containing two pointers: *item.key* points to the comparison key, and *item.data* points to any other data to be associated with that key. (Pointers to types other than character should be cast to pointer-to-character.) *Action* is a member of an enumeration type `ACTION` indicating the disposition of the entry if it cannot be found in the table. `ENTER` indicates that the item should be inserted in the table at an appropriate point. `FIND` indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a `NULL` pointer.

*Hcreate* allocates sufficient space for the table, and must be called before *hsearch* is used. *nel* is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

*Hdestroy* destroys the search table, and may be followed by another call to *hcreate*.

## NOTES

*Hsearch* uses *open addressing* with a *multiplicative* hash function. However, its source code has many other options available which the user may select by compiling the *hsearch* source with the following symbols defined to the preprocessor:

- `DIV` Use the *remainder modulo table size* as the hash function instead of the multiplicative algorithm.
- `USCR` Use a User Supplied Comparison Routine for ascertaining table membership. The routine should be named *hcompar* and should behave in a manner similar to *strcmp* (see *string(3C)*).
- `CHAINED` Use a linked list to resolve collisions. If this option is selected, the following other options become available.
  - `START` Place new entries at the beginning of the linked list (default is at the end).
  - `SORTUP` Keep the linked list sorted by key in ascending order.
  - `SORTDOWN` Keep the linked list sorted by key in descending order.

Additionally, there are preprocessor flags for obtaining debugging printout (`-DDEBUG`) and for including a test driver in the calling routine (`-DDRIVER`). The source code should be consulted for further details.

**SEE ALSO**

bsearch(3C), lsearch(3C), string(3C), tsearch(3C).

**DIAGNOSTICS**

*Hsearch* returns a NULL pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.

*Hcreate* returns zero if it cannot allocate sufficient space for the table.

**BUGS**

Only one hash search table may be active at any given time.

**NAME**

hypot — Euclidean distance function

**SYNOPSIS**

```
#include <math.h>
```

```
double hypot (x, y)
```

```
double x, y;
```

**DESCRIPTION**

*Hypot* returns

```
sqrt(x * x + y * y),
```

taking precautions against unwarranted overflows.

**DIAGNOSTICS**

When the correct value would overflow, *hypot* returns HUGE and sets *errno* to ERANGE.

These error-handling procedures may be changed with the function *matherr*(3M).

**SEE ALSO**

*matherr*(3M), *sqrt*(3F).



**NAME**

`index` — return location of Fortran substring

**SYNOPSIS**

**character\*N1** *ch1*

**character\*N2** *ch2*

**integer** *i*

***i* = index(*ch1*, *ch2*)**

**DESCRIPTION**

*Index* returns the location of substring *ch2* in string *ch1*. The value returned is the position at which substring *ch2* starts, or 0 if it is not present in string *ch1*.

**NAME**

*l3tol*, *ltol3* — convert between 3-byte integers and long integers

**SYNOPSIS**

```
void l3tol (lp, cp, n)
long *lp;
char *cp;
int n;
```

```
void ltol3 (cp, lp, n)
char *cp;
long *lp;
int n;
```

**DESCRIPTION**

*L3tol* converts a list of *n* three-byte integers packed into a character string pointed to by *cp* into a list of long integers pointed to by *lp*.

*Ltol3* performs the reverse conversion from long integers (*lp*) to three-byte integers (*cp*).

These functions are useful for file-system maintenance where the block numbers are three bytes long.

**SEE ALSO**

*fs(4)*.

**BUGS**

Because of possible differences in byte ordering, the numerical values of the long integers are machine-dependent.

**NAME**

*ldahread* — read the archive header of a member of an archive file

**SYNOPSIS**

```
#include <stdio.h>
#include <ar.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
int ldahread (ldptr, arhead)
LDFILE *ldptr;
ARCHDR *arhead;
```

**DESCRIPTION**

If *TYPE(ldptr)* is the archive file magic number, *ldahread* reads the archive header of the common object file currently associated with *ldptr* into the area of memory beginning at *arhead*.

*Ldahread* returns **SUCCESS** or **FAILURE**. *Ldahread* will fail if *TYPE(ldptr)* does not represent an archive file, or if it cannot read the archive header.

The program must be loaded with the object file access routine library *libld.a*.

**SEE ALSO**

*ldclose(3X)*, *ldopen(3X)*, *ldfcn(4)*.

**NAME**

*ldclose*, *ldaclose* — close a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
int ldclose (ldptr)
LDFILE *ldptr;
```

```
int ldaclose (ldptr)
LDFILE *ldptr;
```

**DESCRIPTION**

*Ldopen*(3X) and *ldclose* are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of common object files can be processed as if it were a series of simple common object files.

If *TYPE(ldptr)* does not represent an archive file, *ldclose* will close the file and free the memory allocated to the LDFILE structure associated with *ldptr*. If *TYPE(ldptr)* is the magic number of an archive file, and if there are any more files in the archive, *ldclose* will reinitialize *OFFSET(ldptr)* to the file address of the next archive member and return FAILURE. The LDFILE structure is prepared for a subsequent *ldopen*(3X). In all other cases, *ldclose* returns SUCCESS.

*Ldaclose* closes the file and frees the memory allocated to the LDFILE structure associated with *ldptr* regardless of the value of *TYPE(ldptr)*. *Ldaclose* always returns SUCCESS. The function is often used in conjunction with *ldaopen*.

The program must be loaded with the object file access routine library *libld.a*.

**SEE ALSO**

*fclose*(3S), *ldopen*(3X), *ldfcn*(4).

## NAME

ldfhread — read the file header of a common object file

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
int ldfhread (ldptr, filehead)
LDFILE *ldptr;
FILHDR *filehead;
```

## DESCRIPTION

*Ldfhread* reads the file header of the common object file currently associated with *ldptr* into the area of memory beginning at *filehead*.

*Ldfhread* returns SUCCESS or FAILURE. *Ldfhread* will fail if it cannot read the file header.

In most cases the use of *ldfhread* can be avoided by using the macro **HEADER(*ldptr*)** defined in *ldfcn.h* (see *ldfcn(4)*). The information in any field, *fieldname*, of the file header may be accessed using **HEADER(*ldptr*).*fieldname***.

The program must be loaded with the object file access routine library *libld.a*.

## SEE ALSO

*ldclose(3X)*, *ldopen(3X)*, *ldfcn(4)*.

## NAME

*ldread*, *ldlinit*, *ldlitem* — manipulate line number entries of a common object file function

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <linenum.h>
#include <ldfcn.h>
```

```
int ldread(ldptr, fcnindx, linenum, linent)
```

```
LDFILE *ldptr;
long fcnindx;
unsigned short linenum;
LINENO linent;
```

```
int ldlinit(ldptr, fcnindx)
```

```
LDFILE *ldptr;
long fcnindx;
```

```
int ldlitem(ldptr, linenum, linent)
```

```
LDFILE *ldptr;
unsigned short linenum;
LINENO linent;
```

## DESCRIPTION

*Ldread* searches the line number entries of the common object file currently associated with *ldptr*. *Ldread* begins its search with the line number entry for the beginning of a function and confines its search to the line numbers associated with a single function. The function is identified by *fcnindx*, the index of its entry in the object file symbol table. *Ldread* reads the entry with the smallest line number equal to or greater than *linenum* into *linent*.

*Ldlinit* and *ldlitem* together perform exactly the same function as *ldread*. After an initial call to *ldread* or *ldlinit*, *ldlitem* may be used to retrieve a series of line number entries associated with a single function. *Ldlinit* simply locates the line number entries for the function identified by *fcnindx*. *Ldlitem* finds and reads the entry with the smallest line number equal to or greater than *linenum* into *linent*.

*Ldread*, *ldlinit*, and *ldlitem* each return either SUCCESS or FAILURE. *Ldread* will fail if there are no line number entries in the object file, if *fcnindx* does not index a function entry in the symbol table, or if it finds no line number equal to or greater than *linenum*. *Ldlinit* will fail if there are no line number entries in the object file or if *fcnindx* does not index a function entry in the symbol table. *Ldlitem* will fail if it finds no line number equal to or greater than *linenum*.

The programs must be loaded with the object file access routine library *libld.a*.

## SEE ALSO

*ldclose*(3X), *ldopen*(3X), *ldtbindx*(3X), *ldfcn*(4).

**NAME**

`ldlsæk,ldnlseek` — seek to line number entries of a section of a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldlsæk (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnlseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

**DESCRIPTION**

*Ldlsæk* seeks to the line number entries of the section specified by *sectindx* of the common object file currently associated with *ldptr*.

*Ldnlseek* seeks to the line number entries of the section specified by *sectname*.

*Ldlsæk* and *ldnlseek* return **SUCCESS** or **FAILURE**. *Ldlsæk* will fail if *sectindx* is greater than the number of sections in the object file; *ldnlseek* will fail if there is no section name corresponding with *\*sectname*. Either function will fail if the specified section has no line number entries or if it cannot seek to the specified line number entries.

Note that the first section has an index of *one*.

The program must be loaded with the object file access routine library `libld.a`.

**SEE ALSO**

`ldclose(3X)`, `ldopen(3X)`, `ldshread(3X)`, `ldfcn(4)`.

**NAME**

*ldohseek* — seek to the optional file header of a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldohseek (ldptr)
LDFILE *ldptr;
```

**DESCRIPTION**

*Ldohseek* seeks to the optional file header of the common object file currently associated with *ldptr*.

*Ldohseek* returns **SUCCESS** or **FAILURE**. *Ldohseek* will fail if the object file has no optional header or if it cannot seek to the optional header.

The program must be loaded with the object file access routine library *libld.a*.

**SEE ALSO**

*ldclose(3X)*, *ldopen(3X)*, *ldhread(3X)*, *ldfcn(4)*.



## NAME

*ldopen*, *ldaopen* — open a common object file for reading

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

LDFILE *ldopen (filename, ldptr)
char *filename;
LDFILE *ldptr;

LDFILE *ldaopen (filename, oldptr)
char *filename;
LDFILE *oldptr;
```

## DESCRIPTION

*Ldopen* and *ldclose*(3X) are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of common object files can be processed as if it were a series of simple common object files.

If *ldptr* has the value NULL, then *ldopen* will open *filename* and allocate and initialize the LDFILE structure, and return a pointer to the structure to the calling program.

If *ldptr* is valid and if *TYPE(ldptr)* is the archive magic number, *ldopen* will reinitialize the LDFILE structure for the next archive member of *filename*.

*Ldopen* and *ldclose* are designed to work in concert. *Ldclose* will return FAILURE only when *TYPE(ldptr)* is the archive magic number and there is another file in the archive to be processed. Only then should *ldopen* be called with the current value of *ldptr*. In all other cases, in particular whenever a new *filename* is opened, *ldopen* should be called with a NULL *ldptr* argument.

The following is a prototype for the use of *ldopen* and *ldclose*.

```
/* for each filename to be processed */
ldptr = NULL;
do
    if ( (ldptr = ldopen(filename, ldptr)) != NULL )
    {
        /* check magic number */
        /* process the file */
    }
} while (ldclose(ldptr) == FAILURE);
```

If the value of *oldptr* is not NULL, *ldaopen* will open *filename* anew and allocate and initialize a new LDFILE structure, copying the *TYPE*, *OFFSET*, and *HEADER* fields from *oldptr*. *Ldaopen* returns a pointer to the new LDFILE structure. This new pointer is independent of the old pointer, *oldptr*. The two pointers may be used concurrently to read separate parts of the object file. For example, one pointer may be used to step sequentially through the relocation information, while the other is used to read indexed symbol table entries.

Both *ldopen* and *ldaopen* open *filename* for reading. Both functions return NULL if *filename* cannot be opened, or if memory for the LDFILE structure cannot be allocated. A successful open does not insure that the given file is a common object file or an archived object file.

.The program must be loaded with the object file access routine library **libld.a**.

**SEE ALSO**

**fopen(3S)**, **ldclose(3X)**, **ldfcn(4)**.

**NAME**

*ldrseek*, *ldnrseek* — seek to relocation entries of a section of a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldrseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnrseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

**DESCRIPTION**

*Ldrseek* seeks to the relocation entries of the section specified by *sectindx* of the common object file currently associated with *ldptr*.

*Ldnrseek* seeks to the relocation entries of the section specified by *sectname*.

*Ldrseek* and *ldnrseek* return SUCCESS or FAILURE. *Ldrseek* will fail if *sectindx* is greater than the number of sections in the object file; *ldnrseek* will fail if there is no section name corresponding with *sectname*. Either function will fail if the specified section has no relocation entries or if it cannot seek to the specified relocation entries.

Note that the first section has an index of *one*.

The program must be loaded with the object file access routine library *libld.a*.

**SEE ALSO**

*ldclose*(3X), *ldopen*(3X), *ldshread*(3X), *ldfcn*(4).

**NAME**

`ldshread`, `ldnshread` — read an indexed/named section header of a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <scnhdr.h>
#include <ldfcn.h>

int ldshread (ldptr, sectindx, secthead)
LDFILE *ldptr;
unsigned short sectindx;
SCNHDR *secthead;

int ldnshread (ldptr, sectname, secthead)
LDFILE *ldptr;
char sectname;
SCNHDR *secthead;
```

**DESCRIPTION**

*Ldshread* reads the section header specified by *sectindx* of the common object file currently associated with *ldptr* into the area of memory beginning at *secthead*.

*Ldnshread* reads the section header specified by *sectname* into the area of memory beginning at *secthead*.

*Ldshread* and *ldnshread* return SUCCESS or FAILURE. *Ldshread* will fail if *sectindx* is greater than the number of sections in the object file; *ldnshread* will fail if there is no section name corresponding with *sectname*. Either function will fail if it cannot read the specified section header.

Note that the first section header has an index of *one*.

The program must be loaded with the object file access routine library `libld.a`.

**SEE ALSO**

`ldclose(3X)`, `ldopen(3X)`, `ldfcn(4)`.

**NAME**

`ldsseek`, `ldnsseek` — seek to an indexed/named section of a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldsseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnsseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

**DESCRIPTION**

*Ldsseek* seeks to the section specified by *sectindx* of the common object file currently associated with *ldptr*.

*Ldnsseek* seeks to the section specified by *sectname*.

*Ldsseek* and *ldnsseek* return SUCCESS or FAILURE. *Ldsseek* will fail if *sectindx* is greater than the number of sections in the object file; *ldnsseek* will fail if there is no section name corresponding with *sectname*. Either function will fail if there is no section data for the specified section or if it cannot seek to the specified section.

Note that the first section has an index of *one*.

The program must be loaded with the object file access routine library `libld.a`.

**SEE ALSO**

`ldclose(3X)`, `ldopen(3X)`, `ldhread(3X)`, `ldfcn(4)`.

**NAME**

*ldtbindex* — compute the index of a symbol table entry of a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

long ldtbindex (ldptr)
LDFILE *ldptr;
```

**DESCRIPTION**

*Ldtbindex* returns the (**long**) index of the symbol table entry at the current position of the common object file associated with *ldptr*.

The index returned by *ldtbindex* may be used in subsequent calls to *ldtbread(3X)*. However, since *ldtbindex* returns the index of the symbol table entry that begins at the current position of the object file, if *ldtbindex* is called immediately after a particular symbol table entry has been read, it will return the the index of the next entry.

*Ldtbindex* will fail if there are no symbols in the object file, or if the object file is not positioned at the beginning of a symbol table entry.

Note that the first symbol in the symbol table has an index of *zero*.

The program must be loaded with the object file access routine library **libld.a**.

**SEE ALSO**

*ldclose(3X)*, *ldopen(3X)*, *ldtbread(3X)*, *ldtbseek(3X)*, *ldfcn(4)*.

**NAME**

*ldtbread* — read an indexed symbol table entry of a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldtbread (ldptr, symindex, symbol)
LDFILE *ldptr;
long symindex;
SYMENT *symbol;
```

**DESCRIPTION**

*Ldtbread* reads the symbol table entry specified by *symindex* of the common object file currently associated with *ldptr* into the area of memory beginning at *symbol*.

*Ldtbread* returns **SUCCESS** or **FAILURE**. *Ldtbread* will fail if *symindex* is greater than the number of symbols in the object file, or if it cannot read the specified symbol table entry.

Note that the first symbol in the symbol table has an index of *zero*.

The program must be loaded with the object file access routine library *libld.a*.

**SEE ALSO**

*ldclose(3X)*, *ldopen(3X)*, *ldtseek(3X)*, *ldfcn(4)*.

**NAME**

ldtbseek — seek to the symbol table of a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldtbseek (ldptr)
LDFILE *ldptr;
```

**DESCRIPTION**

*Ldtbseek* seeks to the symbol table of the object file currently associated with *ldptr*.

*Ldtbseek* return SUCCESS or FAILURE. *Ldtbseek* will fail if the symbol table has been stripped from the object file, or if it cannot seek to the symbol table.

The program must be loaded with the object file access routine library *libld.a*.

**SEE ALSO**

ldclose(3X), ldopen(3X), ldtbread(3X), ldfcn(4).



**NAME**

len — return length of Fortran string

**SYNOPSIS**

character\*N ch

integer i

i = len(ch)

**DESCRIPTION**

*Len* returns the length of string *ch*.

**NAME**

log, alog, dlog, clog — Fortran natural logarithm intrinsic function

**SYNOPSIS**

real r1, r2  
double precision dp1, dp2  
complex cx1, cx2  
r2 = alog(r1)  
r2 = log(r1)  
dp2 = dlog(dp1)  
dp2 = log(dp1)  
cx2 = clog(cx1)  
cx2 = log(cx1)

**DESCRIPTION**

*Alog* returns the real natural logarithm of its real argument. *Dlog* returns the double-precision natural logarithm of its double-precision argument. *Clog* returns the complex logarithm of its complex argument. The generic function *log* becomes a call to *alog*, *dlog*, or *clog* depending on the type of its argument.

**SEE ALSO**

exp(3M).

**NAME**

log10,alog10,dlog10 — Fortran common logarithm intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
r2 = alog10(r1)
r2 = log10(r1)
dp2 = dlog10(dp1)
dp2 = log10(dp1)
```

**DESCRIPTION**

*Alog10* returns the real common logarithm of its real argument. *Dlog* returns the double-precision common logarithm of its double-precision argument. The generic function *log* becomes a call to *alog* or *dlog* depending on the type of its argument.

**SEE ALSO**

exp(3M).

**NAME**

logname — return login name of user

**SYNOPSIS**

char \*logname( )

**DESCRIPTION**

*Logname* returns a pointer to the null-terminated login name; it extracts the \$LOGNAME variable from the user's environment.

This routine is kept in /lib/libPW.a.

**FILES**

/etc/profile

**SEE ALSO**

env(1), login(1), profile(4), environ(5).

**BUGS**

The return values point to static data whose content is overwritten by each call.

This method of determining a login name is subject to forgery.

**NAME**

lsearch — linear search and update

**SYNOPSIS**

```
char *lsearch ((char *)key, (char *)base, nelp, sizeof(*key), compar)
unsigned *nelp;
int (*compar)( );
```

**DESCRIPTION**

*Lsearch* is a linear search routine generalized from Knuth (6.1) Algorithm S. It returns a pointer into a table indicating where a datum may be found. If the datum does not occur, it is added at the end of the table. *Key* points to the datum to be sought in the table. *Base* points to the first element in the table. *Nelp* points to an integer containing the current number of elements in the table. The integer is incremented if the datum is added to the table. *Compar* is the name of the comparison function which the user must supply (*strcmp*, for example). It is called with two arguments that point to the elements being compared. The function must return zero if the elements are equal and non-zero otherwise.

**NOTES**

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

**SEE ALSO**

bsearch(3C), hsearch(3C), tsearch(3C).

**BUGS**

Undefined results can occur if there is not enough room in the table to add a new item.

## NAME

malloc, free, realloc, calloc — main memory allocator

## SYNOPSIS

```
char *malloc (size)
unsigned size;

void free (ptr)
char *ptr;

char *realloc (ptr, size)
char *ptr;
unsigned size;

char *calloc (nelem, elsize)
unsigned nelem, elsize;
```

## DESCRIPTION

*Malloc* and *free* provide a simple general-purpose memory allocation package. *Malloc* returns a pointer to a block of at least *size* bytes suitably aligned for any use.

The argument to *free* is a pointer to a block previously allocated by *malloc*; after *free* is performed this space is made available for further allocation, but its contents are left undisturbed.

Undefined results will occur if the space assigned by *malloc* is overrun or if some random number is handed to *free*.

*Malloc* allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls *sbrk* (see *brk(2)*) to get more memory from the system when there is no suitable space already free.

*Realloc* changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If no free block of *size* bytes is available in the storage arena, then *realloc* will ask *malloc* to enlarge the arena by *size* bytes and will then move the data to the new space.

*Realloc* also works if *ptr* points to a block freed since the last call of *malloc*, *realloc*, or *calloc*; thus sequences of *free*, *malloc* and *realloc* can exploit the search strategy of *malloc* to do storage compaction.

*Calloc* allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

## DIAGNOSTICS

*Malloc*, *realloc* and *calloc* return a NULL pointer if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. When this happens the block pointed to by *ptr* may be destroyed.

## NOTE

Search time increases when many objects have been allocated; that is, if a program allocates but never frees, then each successive allocation takes longer.

## NAME

`matherr` — error-handling function

## SYNOPSIS

```
#include <math.h>

int matherr (x)
struct exception *x;
```

## DESCRIPTION

*Matherr* is invoked by functions in the Math Library when errors are detected. Users may define their own procedures for handling errors by including a function named *matherr* in their programs. *Matherr* must be of the form described above. A pointer to the exception structure *x* will be passed to the user-supplied *matherr* function when an error occurs. This structure, which is defined in the `<math.h>` header file, is as follows:

```
struct exception {
    int type;
    char *name;
    double arg1, arg2, retval;
};
```

The element *type* is an integer describing the type of error that has occurred, from the following list of constants (defined in the header file):

DOMAIN	domain error
SING	singularity
OVERFLOW	overflow
UNDERFLOW	underflow
TLOSS	total loss of significance
PLOSS	partial loss of significance

The element *name* points to a string containing the name of the function that had the error. The variables *arg1* and *arg2* are the arguments to the function that had the error. *Retval* is a double that is returned by the function having the error. If it supplies a return value, the user's *matherr* must return non-zero. If the default error value is to be returned, the user's *matherr* must return 0.

If *matherr* is not supplied by the user, the default error-handling procedures, described with the math functions involved, will be invoked upon error. These procedures are also summarized in the table below. In every case, *errno* is set to non-zero and the program continues.

## EXAMPLE

```
matherr(x)
register struct exception *x;
{
    switch (x->type) {
    case DOMAIN:
    case SING: /* print message and abort */
        fprintf(stderr, "domain error in %s\n", x->name);
        abort();
    case OVERFLOW:
        if (!strcmp("exp", x->name)) {
            /* if exp, print message, return the argument */
            fprintf(stderr, "exp of %f\n", x->arg1);
            x->retval = x->arg1;
        } else if (!strcmp("sinh", x->name)) {
            /* if sinh, set errno, return 0 */
```

```

        errno = ERANGE;
        x->retval = 0;
    } else
        /* otherwise, return HUGE */
        x->retval = HUGE;
    break;
case UNDERFLOW:
    return (0); /* execute default procedure */
case TLOSS:
case PLOSS:
    /* print message and return 0 */
    fprintf(stderr, "loss of significance in %s\n", x->name);
    x->retval = 0;
    break;
}
return (1);
}

```

DEFAULT ERROR HANDLING PROCEDURES						
	<i>Types of Errors</i>					
	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS	PLOSS
BESSEL: y0, y1, yn (neg. no.)	— M, -H	—	H —	0 —	—	* —
EXP:	—	—	H	0	—	—
POW: (neg.)**(non- int.), 0**0	— M, 0	—	H —	0 —	—	—
LOG: log(0): log(neg.):	— M, -H	M, -H —	— —	— —	— —	— —
SQRT:	— M, 0	—	—	—	—	—
GAMMA:	—	M, H	—	—	—	—
HYPOT:	—	—	H	—	—	—
SINH, COSH:	—	—	H	—	—	—
SIN, COS:	—	—	—	—	M, 0	M, *
TAN:	—	—	H	—	0	*
ACOS, ASIN:	M, 0	—	—	—	—	—

ABBREVIATIONS	
*	As much as possible of the value is returned.
M	Message is printed.
H	HUGE is returned.
-H	-HUGE is returned.
0	0 is returned.

3



## NAME

`max`, `max0`, `amax0`, `max1`, `amax1`, `dmax1` — Fortran maximum-value functions

## SYNOPSIS

```
integer i, j, k, l
real a, b, c, d
double precision dp1, dp2, dp3

l = max(i, j, k)
c = max(a, b)
dp = max(a, b, c)
k = max0(i, j)
a = amax0(i, j, k)
i = max1(a, b)
d = amax1(a, b, c)
dp3 = dmax1(dp1, dp2)
```

## DESCRIPTION

The maximum-value functions return the largest of their arguments (of which there may be any number). *Max* is the generic form which can be used for all data types and takes its return type from that of its arguments (which must all be of the same type). *Max0* returns the integer form of the maximum value of its integer arguments; *amax0*, the real form of its integer arguments; *max1*, the integer form of its real arguments; *amax1*, the real form of its real arguments; and *dmax1*, the double-precision form of its double-precision arguments.

## SEE ALSO

`min(3F)`.

**NAME**

mclock — return Fortran time accounting

**SYNOPSIS**

integer i

i = mclock( )

**DESCRIPTION**

*Mclock* returns time accounting information about the current process and its child processes. The value returned is the sum of the current process's user time and the user and system times of all child processes.

**SEE ALSO**

times(2), clock(3C), system(3F).

## NAME

memccpy, memchr, memcmp, memcpy, memset — memory operations

## SYNOPSIS

```
#include <memory.h>

char *memccpy (s1, s2, c, n)
char *s1, *s2;
int c, n;

char *memchr (s, c, n)
char *s;
int c, n;

int memcmp (s1, s2, n)
char *s1, *s2;
int n;

char *memcpy (s1, s2, n)
char *s1, *s2;
int n;

char *memset (s, c, n)
char *s;
int c, n;
```

## DESCRIPTION

These functions operate efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

*Memccpy* copies characters from memory area *s2* into *s1*, stopping after the first occurrence of character *c* has been copied, or after *n* characters have been copied, whichever comes first. It returns a pointer to the character after the copy of *c* in *s1*, or a NULL pointer if *c* was not found in the first *n* characters of *s2*.

*Memchr* returns a pointer to the first occurrence of character *c* in the first *n* characters of memory area *s*, or a NULL pointer if *c* does not occur.

*Memcmp* compares its arguments, looking at the first *n* characters only, and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*.

*Memcpy* copies *n* characters from memory area *s2* to *s1*. It returns *s1*.

*Memset* sets the first *n* characters in memory area *s* to the value of character *c*. It returns *s*.

## NOTE

For user convenience, all these functions are declared in the optional *<memory.h>* header file.

## BUGS

*Memcmp* uses native character comparison, which is signed on PDP-11s, unsigned on other machines.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

## NAME

*min*, *min0*, *amin0*, *min1*, *amin1*, *dmin1* — Fortran minimum-value functions

## SYNOPSIS

```
integer i, j, k, l
real a, b, c, d
double precision dp1, dp2, dp3

l = min(i, j, k)
c = min(a, b)
dp = min(a, b, c)
k = min0(i, j)
a = amin0(i, j, k)
i = min1(a, b)
d = amin1(a, b, c)
dp3 = dmin1(dp1, dp2)
```

## DESCRIPTION

The minimum-value functions return the minimum of their arguments (of which there may be any number). *Min* is the generic form which can be used for all data types and takes its return type from that of its arguments (which must all be of the same type). *Min0* returns the integer form of the minimum value of its integer arguments; *amin0*, the real form of its integer arguments; *min1*, the integer form of its real arguments; *amin1*, the real form of its real arguments; and *dmin1*, the double-precision form of its double-precision arguments.

## SEE ALSO

*max*(3F).

**NAME**

`mktemp` — make a unique file name

**SYNOPSIS**

```
char *mktemp (template)
char *template;
```

**DESCRIPTION**

*Mktemp* replaces the contents of the string pointed to by *template* by a unique file name, and returns the address of *template*. The string in *template* should look like a file name with six trailing Xs; *mktemp* will replace the Xs with a letter and the current process ID. The letter will be chosen so that the resulting name does not duplicate an existing file.

**SEE ALSO**

`getpid(2)`, `tmpfile(3S)`, `tmpnam(3S)`.

**BUGS**

It is possible to run out of letters.

**NAME**

**mod, amod, dmod** — Fortran remaindering intrinsic functions

**SYNOPSIS**

**integer i, j, k**  
**real r1, r2, r3**  
**double precision dp1, dp2, dp3**  
**k = mod(i, j)**  
**r3 = amod(r1, r2)**  
**r3 = mod(r1, r2)**  
**dp3 = dmod(dp1, dp2)**  
**dp3 = mod(dp1, dp2)**

**DESCRIPTION**

*Mod* returns the integer remainder of its first argument divided by its second argument. *Amod* and *dmod* return, respectively, the real and double-precision whole number remainder of the integer division of their two arguments. The generic version *mod* will return the data type of its arguments.

**NAME**

monitor — prepare execution profile

**SYNOPSIS**

```
void monitor (lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)( ), (*highpc)( );
short *buffer;
int bufsize, nfunc;
```

**DESCRIPTION**

An executable program created by `cc -p` automatically includes calls for *monitor* with default parameters; *monitor* needn't be called explicitly except to gain fine control over profiling.

*Monitor* is an interface to *profil(2)*. *Lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* short integers. *Monitor* arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of *lowpc* and the highest is just below *highpc*. *Lowpc* may not equal 0 for this use of *monitor*. At most *nfunc* call counts can be kept; only calls of functions compiled with the profiling option `-p` of *cc(1)* are recorded. (The C Library and Math Library supplied when `cc -p` is used also have call counts recorded.) For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use

```
extern etext;
...
monitor ((int (*)())2, etext, buf, bufsize, nfunc);
```

*Ettext* lies just above all the program text; see *end(3C)*.

To stop execution monitoring and write the results on the file **mon.out**, use

```
monitor ((int (*)())NULL, 0, 0, 0, 0);
```

*Prof(1)* can then be used to examine the results.

**FILES**

mon.out

**SEE ALSO**

*cc(1)*, *prof(1)*, *profil(2)*, *end(3C)*.

**NAME**

nlist — get entries from name list

**SYNOPSIS**

```
#include <a.out.h>

int nlist (file-name, nl)
char *file-name;
struct nlist *nl[ ];
```

**DESCRIPTION**

*Nlist* examines the name list in the executable file whose name is pointed to by *file-name*, and selectively extracts a list of values and puts them in the array of nlist structures pointed to by *nl*. The name list *nl* consists of an array of structures containing names of variables, types and values. The list is terminated with a null name; that is, a null string is in the name position of the structure. Each variable name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to 0. See *a.out(4)* for a discussion of the symbol table structure.

This subroutine is useful for examining the system name list kept in the file */unix*. In this way programs can obtain system addresses that are up to date.

**SEE ALSO**

*a.out(4)*.

**DIAGNOSTICS**

All type entries are set to 0 if the file cannot be read or if it doesn't contain a valid name list.

*Nlist* returns -1 upon error; otherwise it returns 0.



**NAME**

`perror`, `errno`, `sys_errlist`, `sys_nerr` -- system error messages

**SYNOPSIS**

```
void perror (s)
char *s;
extern int errno;
extern char *sys_errlist[ ];
extern int sys_nerr;
```

**DESCRIPTION**

*Perror* produces a message on the standard error output, describing the last error encountered during a call to a system or library function. The argument string *s* is printed first, then a colon and a blank, then the message and a new-line. To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable *errno*, which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the array of message strings *sys\_errlist* is provided; *errno* can be used as an index in this table to get the message string without the new-line. *sys\_nerr* is the largest message number provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

**SEE ALSO**

`intro(2)`.

## NAME

plot — graphics interface subroutines

## SYNOPSIS

```

openpl ( )
erase ( )
label (s)
char *s;
line (x1, y1, x2, y2)
int x1, y1, x2, y2;
circle (x, y, r)
int x, y, r;
arc (x, y, x0, y0, x1, y1)
int x, y, x0, y0, x1, y1;
move (x, y)
int x, y;
cont (x, y)
int x, y;
point (x, y)
int x, y;
linemod (s)
char *s;
space (x0, y0, x1, y1)
int x0, y0, x1, y1;
closepl ( )

```

## DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. *Space* must be used before any of these functions to declare the amount of space necessary. See *plot(4)*. *Openpl* must be used before any of the others to open the device for writing. *Closepl* flushes the output.

*Circle* draws a circle of radius *r* with center at the point  $(x, y)$ .

*Arc* draws an arc of a circle with center at the point  $(x, y)$  between the points  $(x0, y0)$  and  $(x1, y1)$ .

String arguments to *label* and *linemod* are terminated by nulls and do not contain new-lines.

See *plot(4)* for a description of the effect of the remaining functions.

The library files listed below provide several flavors of these routines.

## FILES

/usr/lib/libplot.a	produces output for <i>tplot(1G)</i> filters
/usr/lib/lib300.a	for DASI 300
/usr/lib/lib300s.a	for DASI 300s
/usr/lib/lib450.a	for DASI 450
/usr/lib/lib4014.a	for Tektronix 4014

## WARNINGS

In order to compile a program containing these functions in *file.c* it is necessary to use “*cc file.c -lplot*”.

In order to execute it, it is necessary to use “*a.out | tplot*”.

The above routines use `<stdio.h>`, which causes them to increase the size of programs, not otherwise using standard I/O, more than might be expected.

**SEE ALSO**

`graph(1G)`, `stat(1G)`, `tplot(1G)`, `plot(4)`.

**NAME**

*popen*, *pclose* — initiate pipe to/from a process

**SYNOPSIS**

```
#include <stdio.h>

FILE *popen (command, type)
char *command, *type;

int pclose (stream)
FILE *stream;
```

**DESCRIPTION**

The arguments to *popen* are pointers to null-terminated strings containing, respectively, a shell command line and an I/O mode, either *r* for reading or *w* for writing. *Popen* creates a pipe between the calling program and the command to be executed. The value returned is a stream pointer such that one can write to the standard input of the command, if the I/O mode is *w*, by writing to the file *stream*; and one can read from the standard output of the command, if the I/O mode is *r*, by reading from the file *stream*.

A stream opened by *popen* should be closed by *pclose*, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type *r* command may be used as an input filter and a type *w* as an output filter.

**SEE ALSO**

*pipe(2)*, *wait(2)*, *fclose(3S)*, *fopen(3S)*, *system(3S)*.

**DIAGNOSTICS**

*Popen* returns a NULL pointer if files or processes cannot be created, or if the shell cannot be accessed.

*Pclose* returns  $-1$  if *stream* is not associated with a “*popen ed*” command.

**BUGS**

If the original and “*popen ed*” processes concurrently read or write a common file, neither should use buffered I/O, because the buffering gets all mixed up. Problems with an output filter may be forestalled by careful buffer flushing, e.g. with *fflush*; see *fclose(3S)*.

## NAME

printf, fprintf, sprintf — print formatted output

## SYNOPSIS

```
#include <stdio.h>

int printf (format [ , arg ] ... )
char *format;

int fprintf (stream, format [ , arg ] ... )
FILE *stream;
char *format;

int sprintf (s, format [ , arg ] ... )
char *s, format;
```

## DESCRIPTION

*Printf* places output on the standard output stream *stdout*. *Fprintf* places output on the named output *stream*. *Sprintf* places “output”, followed by the null character (\0) in consecutive bytes starting at *\*s*; it is the user’s responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the \0 in the case of *sprintf*), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its *args* under control of the *format*. The *format* is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more *args*. The results are undefined if there are insufficient *args* for the format. If the format is exhausted while *args* remain, the excess *args* are simply ignored.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag (see below) has been given) to the field width;

A *precision* that gives the minimum number of digits to appear for the *d*, *o*, *u*, *x*, or *X* conversions, the number of digits to appear after the decimal point for the *e* and *f* conversions, the maximum number of significant digits for the *g* conversion, or the maximum number of characters to be printed from a string in *s* conversion. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero.

An optional *l* specifying that a following *d*, *o*, *u*, *x*, or *X* conversion character applies to a long integer *arg*.

A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (\*) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *args* specifying field width or precision must appear *before* the *arg* (if any) to be converted.

The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field.
- + The result of a signed conversion will always begin with a sign (+ or -).
- blank If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.
- # This flag specifies that the value is to be converted to an "alternate form." For c, d, s, and u conversions, the flag has no effect. For o conversion, it increases the precision to force the first digit of the result to be a zero. For x (X) conversion, a non-zero result will have 0x (0X) prefixed to it. For e, E, f, g, and G conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For g and G conversions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

- d,o,u,x,X The integer *arg* is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation (x and X), respectively; the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of zero is a null string.
- f The float or double *arg* is converted to decimal notation in the style "[−]ddd.ddd", where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are output; if the precision is explicitly 0, no decimal point appears.
- e,E The float or double *arg* is converted in the style "[−]d.ddde±dd", where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The E format code will produce a number with E instead of e introducing the exponent. The exponent always contains at least two digits.
- g,G The float or double *arg* is printed in style f or e (or in style E in the case of a G format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style e will be used only if the exponent resulting from the conversion is less than -4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.
- c The character *arg* is printed.
- s The *arg* is taken to be a string (character pointer) and characters from the string are printed until a null character (\0) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. If the string pointer *arg* has the value zero, the result is undefined. A *null arg* will yield undefined results.
- % Print a %; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is

simply expanded to contain the conversion result. Characters generated by *printf* and *fprintf* are printed as if *putc*(3S) had been called.

**EXAMPLES**

To print a date and time in the form "Sunday, July 3, 10:02", where *weekday* and *month* are pointers to null-terminated strings:

```
printf("%s, %s %d, %.2d:%.2d", weekday, month, day, hour, min);
```

To print  $\pi$  to 5 decimal places:

```
printf("pi = %.5f", 4*atan(1.0));
```

**SEE ALSO**

ecvt(3C), putc(3S), scanf(3S), stdio(3S).

## NAME

putc, putchar, fputc, putw — put character or word on a stream

## SYNOPSIS

```
#include <stdio.h>
int putc (c, stream)
char c;
FILE *stream;

int putchar (c)
char c;

int fputc (c, stream)
char c;
FILE *stream;

int putw (w, stream)
int w;
FILE *stream;
```

## DESCRIPTION

*putc* writes the character *c* onto the output *stream* (at the position where the file pointer, if defined, is pointing). *Putchar(c)* is defined as *putc(c, stdout)*. *putc* and *putchar* are macros.

*Fputc* behaves like *putc*, but is a function rather than a macro. *Fputc* runs more slowly than *putc*, but takes less space per invocation.

*Putw* writes the word (i.e. integer) *w* to the output *stream* (at the position at which the file pointer, if defined, is pointing). The size of a word is the size of an integer and varies from machine to machine. *Putw* neither assumes nor causes special alignment in the file.

Output streams, with the exception of the standard error stream *stderr*, are by default buffered if the output refers to a file and line-buffered if the output refers to a terminal. The standard error output stream *stderr* is by default unbuffered, but use of *freopen* (see *fopen*(3S)) will cause it to become buffered or line-buffered. When an output stream is unbuffered information is queued for writing on the destination file or terminal as soon as written; when it is buffered many characters are saved up and written as a block; when it is line-buffered each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a new-line character is written or terminal input is requested). *Setbuf*(3S) may be used to change the stream's buffering strategy.

## SEE ALSO

*fclose*(3S), *ferror*(3S), *fopen*(3S), *fread*(3S), *printf*(3S), *puts*(3S), *setbuf*(3S).

## DIAGNOSTICS

On success, these functions each return the value they have written. On failure, they return the constant EOF. This will occur if the file *stream* is not open for writing, or if the output file cannot be grown. Because EOF is a valid integer, *ferror*(3S) should be used to detect *putw* errors.

## BUGS

Because it is implemented as a macro, *putc* treats incorrectly a *stream* argument with side effects. In particular, *putc(c, \*f++)*; doesn't work sensibly. *Fputc* should be used instead.

Because of possible differences in word length and byte ordering, files written using *putw* are machine-dependent, and may not be read using *getw* on a different processor. For this reason the use of *putw* should be avoided.



**NAME**

putpwent — write password file entry

**SYNOPSIS**

```
#include <pwd.h>
int putpwent (p, f)
struct passwd *p;
FILE *f;
```

**DESCRIPTION**

*Putpwent* is the inverse of *getpwent*(3C). Given a pointer to a *passwd* structure created by *getpwent* (or *getpwuid* or *getpwnam*), *putpwuid* writes a line on the stream *f* which matches the format of */etc/passwd*.

**DIAGNOSTICS**

*Putpwent* returns non-zero if an error was detected during its operation, otherwise zero.

**WARNING**

The above routine uses *<stdio.h>*, which causes it to increase the size of programs, not otherwise using standard I/O, more than might be expected.

**NAME**

puts, fputs — put a string on a stream

**SYNOPSIS**

```
#include <stdio.h>
```

```
int puts (s)
```

```
char *s;
```

```
int fputs (s, stream)
```

```
char *s;
```

```
FILE *stream;
```

**DESCRIPTION**

*Puts* writes the null-terminated string pointed to by *s*, followed by a new-line character, to the standard output stream *stdout*.

*Fputs* writes the null-terminated string pointed to by *s* to the named output *stream*.

Neither function writes the terminating null character.

**DIAGNOSTICS**

Both routines return EOF on error. This will happen if the routines try to write on a file that has not been opened for writing.

**SEE ALSO**

ferror(3S), fopen(3S), fread(3S), printf(3S), putc(3S).

**NOTES**

*Puts* appends a new-line character while *fputs* does not.

**NAME**

qsort — quicker sort

**SYNOPSIS**

```
void qsort ((char *) base, nel, sizeof (*base), compar)
unsigned int nel;
int (*compar)( );
```

**DESCRIPTION**

*Qsort* is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

*Base* points to the element at the base of the table. *Nel* is the number of elements in the table. *Compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero according as the first argument is to be considered less than, equal to, or greater than the second.

**NOTES**

The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

**SEE ALSO**

sort(1), bsearch(3C), lsearch(3C), string(3C).

**NAME**

*rand*, *srand* — simple random-number generator

**SYNOPSIS**

```
int rand ( )  
void srand (seed)  
unsigned seed;
```

**DESCRIPTION**

*Rand* uses a multiplicative congruential random-number generator with period  $2^{32}$  that returns successive pseudo-random numbers in the range from 0 to  $2^{15}-1$ .

*Srand* can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

**NOTE**

The spectral properties of *rand* leave a great deal to be desired. *Drand48(3C)* provides a much better, though more elaborate, random-number generator.

**SEE ALSO**

*drand48(3C)*.

**NAME**

*srand*, *rand* — Fortran uniform random-number generator

**SYNOPSIS**

```
integer i, j
call srand(i)
j = rand( )
```

**DESCRIPTION**

*Srand* takes its integer argument as the seed of a random-number generator, the values of which are returned through successive invocations of *rand*.

**SEE ALSO**

*rand*(3C).

## NAME

regcmp, regex — compile and execute regular expression

## SYNOPSIS

```
char *regcmp(string1 [, string2, ...], 0)
char *string1, *string2, ...;

char *regex(re, subject[, ret0, ...])
char *re, *subject, *ret0, ...;

extern char *loc1;
```

## DESCRIPTION

*Regcmp* compiles a regular expression and returns a pointer to the compiled form. *Malloc*(3C) is used to create space for the vector. It is the user's responsibility to free unneeded space so allocated. A NULL return from *regcmp* indicates an incorrect argument. *Regcmp*(1) has been written to generally preclude the need for this routine at execution time.

*Regex* executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. *Regex* returns NULL on failure or a pointer to the next unmatched character on success. A global character pointer *loc1* points to where the match began. *Regcmp* and *regex* were mostly borrowed from the editor, *ed*(1); however, the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings.

- [ ] \* . ^ These symbols retain their current meaning.
- \$ Matches the end of the string, \n matches the new-line.
- Within brackets the minus means *through*. For example, [a-z] is equivalent to [abcd...xyz]. The - can appear as itself only if used as the last or first character. For example, the character class expression [ ]- matches the characters ] and -.
- + A regular expression followed by + means *one or more times*. For example, [0-9]+ is equivalent to [0-9][0-9]\*.
- {m} {m,} {m,u} Integer values enclosed in {} indicate the number of times the preceding regular expression is to be applied. *m* is the minimum number and *u* is a number, less than 256, which is the maximum. If only *m* is present (e.g., {m}), it indicates the exact number of times the regular expression is to be applied. {m,} is analogous to {m,infinity}. The plus (+) and star (\*) operations are equivalent to {1,} and {0,} respectively.
- (...)\$n The value of the enclosed regular expression is to be returned. The value will be stored in the (n+1)th argument following the subject argument. At present, at most ten enclosed regular expressions are allowed. *Regex* makes its assignments unconditionally.
- (...) Parentheses are used for grouping. An operator, e.g. \*, +, {}, can work on a single character or a regular expression enclosed in parenthesis. For example, (a\*(cb+))\*\$0.

By necessity, all the above defined symbols are special. They must, therefore, be escaped to be used as themselves.

## EXAMPLES

Example 1:

```
char *cursor, *newcursor, *ptr;
```

```

...
newcursor = regex((ptr = regcmp("\n", 0)), cursor);
free(ptr);

```

This example will match a leading new-line in the subject string pointed at by cursor.

Example 2:

```

char ret0[9];
char *newcursor, *name;

...
name = regcmp("[A-Za-z][A-Za-z0-9_]{0,7})$0", 0);
newcursor = regex(name, "123Testing321", ret0);

```

This example will match through the string "Testing3" and will return the address of the character after the last matched character (cursor+11). The string "Testing3" will be copied to the character array *ret0*.

Example 3:

```

#include "file.i"
char *string, *newcursor;

...
newcursor = regex(name, string);

```

This example applies a precompiled regular expression in *file.i* (see *regcmp(1)*) against *string*.

This routine is kept in */lib/libPW.a*.

#### SEE ALSO

*ed(1)*, *regcmp(1)*, *malloc(3C)*.

#### BUGS

The user program may run out of memory if *regcmp* is called iteratively without freeing the vectors no longer required. The following user-supplied replacement for *malloc(3C)* reuses the same vector saving time and space:

```

/* user's program */
...
malloc(n) {
    static int rebuf[256];
    return rebuf;
}

```

**NAME**

*anint*, *dnint*, *nint*, *idnint* — Fortran nearest integer functions

**SYNOPSIS**

```
integer i
real r1, r2
double precision dp1, dp2

r2 = anint(r1)
i = nint(r1)

dp2 = anint(dp1)
dp2 = dnint(dp1)

i = nint(dp1)
i = idnint(dp1)
```

**DESCRIPTION**

*Anint* returns the nearest whole real number to its real argument (i.e.,  $\text{int}(a+0.5)$  if  $a \geq 0$ ,  $\text{int}(a-0.5)$  otherwise). *Dnint* does the same for its double-precision argument. *Nint* returns the nearest integer to its real argument. *Idnint* is the double-precision version. *Anint* is the generic form of *anint* and *dnint*, performing the same operation and returning the data type of its argument. *Nint* is also the generic form of *idnint*.



## NAME

scanf, fscanf, sscanf — convert formatted input

## SYNOPSIS

```
#include <stdio.h>

int scanf (format [ , pointer ] ... )
char *format;

int fscanf (stream, format [ , pointer ] ... )
FILE *stream;
char *format;

int sscanf (s, format [ , pointer ] ... )
char *s, *format;
```

## DESCRIPTION

*Scanf* reads from the standard input stream *stdin*. *Fscanf* reads from the named input *stream*. *Sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. White-space characters (blanks, tabs, new-lines, or form-feeds) which, except in two cases described below, cause input to be read up to the next non-white-space character.
2. An ordinary character (not %), which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character \*, an optional numerical maximum field width, an optional l or h indicating the size of the receiving variable, and a conversion code.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by \*. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion code indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument should be given. The following conversion codes are legal:

- |   |  |
|---|--|
| % | a single % is expected in the input at this point; no assignment is done.                                  |
| d | a decimal integer is expected; the corresponding argument should be an integer pointer.                    |
| u | an unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer. |
| o | an octal integer is expected; the corresponding argument should be an integer pointer.                     |
| x | a hexadecimal integer is expected; the corresponding argument should be an integer pointer.                |

- e,f,g a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an **E** or an **e**, followed by an optionally signed integer.
- s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating `\0`, which will be added automatically. The input field is terminated by a white-space character.
- c a character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-space character, use `%1s`. If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.
- [ indicates string data and the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters, which we will call the *scanset*, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. The circumflex, (`^`), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters *not* contained in the remainder of the scanset string. There are some conventions used in the construction of the scanset. A range of characters may be represented by the construct *first-last*, thus `[0123456789]` may be expressed `[0-9]`. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The dash will also stand for itself whenever it is the first or the last character in the scanset. To include the right square bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanset, and in this case it will not be syntactically interpreted as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating `\0`, which will be added automatically.

The conversion characters **d**, **u**, **o**, and **x** may be preceded by **l** or **h** to indicate that a pointer to **long** or to **short** rather than to **int** is in the argument list. Similarly, the conversion characters **e**, **f**, and **g** may be preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list.

*Scanf* conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

*Scanf* returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, EOF is returned.

## EXAMPLES

The call:

```
int i; float x; char name[50];
scanf ("%d%f%s", &i, &x, name);
```

with the input line:

25 54.32E-1 thompson

will assign to *i* the value 25, to *x* the value 5.432, and *name* will contain thompson\0. Or:

```
int i; float x; char name[50];
scanf ("%2d%f%*d %[0-9]", &i, &x, name);
```

with input:

56789 0123 56a72

will assign 56 to *i*, 789.0 to *x*, skip 0123, and place the string 56\0 in *name*. The next call to *getchar* (see *getc*(3S)) will return a.

#### SEE ALSO

atof(3C), *getc*(3S), *printf*(3S), *strtol*(3C).

#### NOTE

Trailing white space (including a new-line) is left unread unless matched in the control string.

#### DIAGNOSTICS

These functions return EOF on end of input and a short count for missing or illegal data items.

#### BUGS

The success of literal matches and suppressed assignments is not directly determinable.

**NAME**

setbuf — assign buffering to a stream

**SYNOPSIS**

```
#include <stdio.h>
void setbuf (stream, buf)
FILE *stream;
char *buf;
```

**DESCRIPTION**

*Setbuf* is used after a stream has been opened but before it is read or written. It causes the character array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is a NULL character pointer input/output will be completely unbuffered.

A constant **BUFSIZ**, defined in the `<stdio.h>` header file, tells how big an array is needed:

```
char buf[BUFSIZ];
```

A buffer is normally obtained from *malloc*(3C) at the time of the first *getc* or *putc*(3S) on the file, except that the standard error stream *stderr* is normally not buffered.

Output streams directed to terminals are always line-buffered unless they are unbuffered.

**SEE ALSO**

fopen(3S), getc(3S), malloc(3C), putc(3S).

**NOTE**

A common source of error is allocating buffer space as an “automatic” variable in a code block, and then failing to close the stream in the same block.

**NAME**

setjmp, longjmp — non-local goto

**SYNOPSIS**

```
#include <setjmp.h>
int setjmp (env)
jmp_buf env;
void longjmp (env, val)
jmp_buf env;
int val;
```

**DESCRIPTION**

These functions are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

*Setjmp* saves its stack environment in *env* (whose type, *jmp\_buf*, is defined in the *<setjmp.h>* header file), for later use by *longjmp*. It returns the value 0.

*Longjmp* restores the environment saved by the last call of *setjmp* with the corresponding *env* argument. After *longjmp* is completed program execution continues as if the corresponding call of *setjmp* (which must not itself have returned in the interim) had just returned the value *val*. *Longjmp* cannot cause *setjmp* to return the value 0. If *longjmp* is invoked with a second argument of 0, *setjmp* will return 1. All accessible data have values as of the time *longjmp* was called.

**SEE ALSO**

signal(2).

**WARNING**

If *longjmp* is called when *env* was never primed by a call to *setjmp*, or when the last such call is in a function which has since returned, absolute chaos is guaranteed.

**NAME**

*sign*, *isign*, *dsign* — Fortran transfer-of-sign intrinsic function

**SYNOPSIS**

```
integer i, j, k
real r1, r2, r3
double precision dp1, dp2, dp3

k = isign(i, j)
k = sign(i, j)

r3 = sign(r1, r2)

dp3 = dsign(dp1, dp2)
dp3 = sign(dp1, dp2)
```

**DESCRIPTION**

*Isign* returns the magnitude of its first argument with the sign of its second argument. *Sign* and *dsign* are its real and double-precision counterparts, respectively. The generic version is *sign* and will devolve to the appropriate type depending on its arguments.

**NAME**

signal — specify Fortran action on receipt of a system signal

**SYNOPSIS**

**integer i**  
**external integer intfnc**  
**call signal(i, intfnc)**

**DESCRIPTION**

*Signal* allows a process to specify a function to be invoked upon receipt of a specific signal. The first argument specifies which fault or exception, the second argument the function to be invoked.

**SEE ALSO**

kill(2), signal(2).

**NAME**

sin, dsin, csin — Fortran sine intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2
r2 = sin(r1)
dp2 = dsin(dp1)
dp2 = sin(dp1)
cx2 = csin(cx1)
cx2 = sin(cx1)
```

**DESCRIPTION**

*Sin* returns the real sine of its real argument. *Dsin* returns the double-precision sine of its double-precision argument. *Csin* returns the complex sine of its complex argument. The generic *sin* function becomes *dsin* or *csin* as required by argument type.

**SEE ALSO**

trig(3M).



**NAME**

*sinh*, *dsinh* — Fortran hyperbolic sine intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
r2 = sinh(r1)
dp2 = dsinh(dp1)
dp2 = sinh(dp1)
```

**DESCRIPTION**

*Sinh* returns the real hyperbolic sine of its real argument. *Dsinh* returns the double-precision hyperbolic sine of its double-precision argument. The generic form *sinh* may be used to return a double-precision value given a double-precision argument.

**SEE ALSO**

*sinh*(3M).

**NAME**

*sinh*, *cosh*, *tanh* — hyperbolic functions

**SYNOPSIS**

```
#include <math.h>
double sinh (x)
double x;
double cosh (x)
double x;
double tanh (x)
double x;
```

**DESCRIPTION**

*Sinh*, *cosh* and *tanh* return respectively the hyperbolic sine, cosine and tangent of their argument.

**DIAGNOSTICS**

*Sinh* and *cosh* return HUGE when the correct value would overflow, and set *errno* to ERANGE.

These error-handling procedures may be changed with the function *matherr*(3M).

**SEE ALSO**

*matherr*(3M).

**NAME**

sleep — suspend execution for interval

**SYNOPSIS**

**unsigned sleep (seconds)**  
**unsigned seconds;**

**DESCRIPTION**

The current process is suspended from execution for the number of *seconds* specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) Because scheduled wakeups occur at fixed 1-second intervals, (on the second, according to an internal clock) and (2) because any caught signal will terminate the *sleep* following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system. The value returned by *sleep* will be the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested *sleep* time, or premature arousal due to another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling *sleep*; if the *sleep* time exceeds the time till such alarm signal, the process sleeps only until the alarm signal would have occurred, and the caller's alarm catch routine is executed just before the *sleep* routine returns, but if the *sleep* time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening *sleep*.

**SEE ALSO**

alarm(2), pause(2), signal(2).

**NAME**

*sputl*, *sgetl* — access long numeric data in a machine independent fashion.

**SYNOPSIS**

```
sputl ( value, buffer )  
long value;  
char *buffer;  
  
long sgetl ( buffer )  
char *buffer;
```

**DESCRIPTION**

*Sputl*(3X) will take the 4 bytes of the long *value* and place them in memory starting at the address pointed to by *buffer*. The ordering of the bytes is the same across all machines. *Sgetl* will retrieve the 4 bytes in memory starting at the address pointed to by *buffer* and return the long value in the byte ordering of the host machine.

The usage of *sputl*(3X) and *sgetl* in combination provides a machine independent way of storing long numeric data in an ASCII file. The numeric data stored in the portable archive file format (see *ar*(4)) is written and read into/from buffers with *sputl*(3X) and *sgetl* respectively.

A program which uses these functions must be loaded with the object file access routine library *libld.a*.

**SEE ALSO**

*ar*(4).

## NAME

*sqrt*, *dsqrt*, *csqrt* — Fortran square root intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2 = sqrt(r1)

dp2 = dsqrt(dp1)
dp2 = sqrt(dp1)

cx2 = csqrt(cx1)
cx2 = sqrt(cx1)
```

## DESCRIPTION

*Sqrt* returns the real square root of its real argument. *Dsqrt* returns the double-precision square root of its double-precision argument. *Csqrt* returns the complex square root of its complex argument. *Sqrt*, the generic form, will become *dsqrt* or *csqrt* as required by its argument type.

## SEE ALSO

*exp(3M)*.

## NAME

*ssignal*, *gsignal* — software signals

## SYNOPSIS

```
#include <signal.h>
int (*ssignal (sig, action))( )
int sig, (*action)( );
int gsignal (sig)
int sig;
```

## DESCRIPTION

*Ssignal* and *gsignal* implement a software facility similar to *signal(2)*. This facility is used by the Standard C Library to enable users to indicate the disposition of error conditions, and is also made available to users for their own purposes.

Software signals made available to users are associated with integers in the inclusive range 1 through 15. A call to *ssignal* associates a procedure, *action*, with the software signal *sig*; the software signal, *sig*, is raised by a call to *gsignal*. Raising a software signal causes the action established for that signal to be *taken*.

The first argument to *ssignal* is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user defined) *action function* or one of the manifest constants SIG\_DFL (default) or SIG\_IGN (ignore). *Ssignal* returns the action previously established for that signal type; if no action has been established or the signal number is illegal, *ssignal* returns SIG\_DFL.

*Gsignal* raises the signal identified by its argument, *sig*:

If an action function has been established for *sig*, then that action is reset to SIG\_DFL and the action function is entered with argument *sig*. *Gsignal* returns the value returned to it by the action function.

If the action for *sig* is SIG\_IGN, *gsignal* returns the value 1 and takes no other action.

If the action for *sig* is SIG\_DFL, *gsignal* returns the value 0 and takes no other action.

If *sig* has an illegal value or no action was ever specified for *sig*, *gsignal* returns the value 0 and takes no other action.

## NOTES

There are some additional signals with numbers outside the range 1 through 15 which are used by the Standard C Library to indicate error conditions. Thus, some signal numbers outside the range 1 through 15 are legal, although their use may interfere with the operation of the Standard C Library.

**NAME**

stdio — standard buffered input/output package

**SYNOPSIS**

```
#include <stdio.h>
FILE *stdin, *stdout, *stderr;
```

**DESCRIPTION**

The functions described in the entries of sub-class 3S of this manual constitute an efficient, user-level I/O buffering scheme. The in-line macros *getc*(3S) and *putc*(3S) handle characters quickly. The macros *getchar*, *putchar*, and the higher-level routines *fgetc*, *fgets*, *sprintf*, *fputc*, *fputs*, *fread*, *fscanf*, *fwrite*, *gets*, *getw*, *printf*, *puts*, *putw*, and *scanf* all use *getc* and *putc*; they can be freely intermixed.

A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type FILE. *Fopen*(3S) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the <stdio.h> header file and associated with the standard open files:

<b>stdin</b>	standard input file
<b>stdout</b>	standard output file
<b>stderr</b>	standard error file.

A constant NULL (0) designates a nonexistent pointer.

An integer constant EOF (-1) is returned upon end-of-file or error by most integer functions that deal with streams (see the individual descriptions for details).

Any program that uses this package must include the header file of pertinent macro definitions, as follows:

```
#include <stdio.h>
```

The functions and constants mentioned in the entries of sub-class 3S of this manual are declared in that header file and need no further declaration. The constants and the following “functions” are implemented as macros (redeclaration of these names is perilous): *getc*, *getchar*, *putc*, *putchar*, *feof*, *error*, *clearerr*, and *fileno*.

**SEE ALSO**

*open*(2), *close*(2), *lseek*(2), *pipe*(2), *read*(2), *write*(2), *ctermid*(3S), *cuserid*(3S), *fclose*(3S), *ferror*(3S), *fopen*(3S), *fread*(3S), *fseek*(3S), *getc*(3S), *gets*(3S), *popen*(3S), *printf*(3S), *putc*(3S), *puts*(3S), *scanf*(3S), *setbuf*(3S), *system*(3S), *tmpfile*(3S), *tmpnam*(3S), *ungetc*(3S).

**DIAGNOSTICS**

Invalid *stream* pointers will usually cause grave disorder, possibly including program termination. Individual function descriptions describe the possible error conditions.

**NAME**

stdipc — standard interprocess communication package

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(path, id)
char *path;
char id;
```

**DESCRIPTION**

All interprocess communication facilities require the user to supply a key to be used by the *msgget(2)*, *semget(2)* and *shmget(2)* system calls to obtain interprocess communication identifiers. One suggested method for forming a key is to use the *ftok* subroutine described below. Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If some standard is not adhered to, it will be possible for unrelated processes to unintentionally interfere with each other's operation. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

*ftok* returns a key based on *path* and *id* that is usable in subsequent *msgget*, *semget* and *shmget* system calls. *Path* must be the path name of an existing file that is accessible to the process. *Id* is a character which uniquely identifies a project. Note that *ftok* will return the same key for linked files when called with the same *id* and that it will return different keys when called with the same file name but different *ids*.

**SEE ALSO**

intro(2), *msgget(2)*, *semget(2)*, *shmget(2)*.

**DIAGNOSTICS**

*ftok* returns (**key\_t**) **-1** if *path* does not exist or if it is not accessible to the process.

**WARNING**

If the file whose *path* is passed to *ftok* is removed when keys still refer to the file, future calls to *ftok* with the same *path* and *id* will return an error. If the same file is recreated, then *ftok* is likely to return a different key than it did the original time it was called.



## NAME

*strcat*, *strncat*, *strcmp*, *strncmp*, *strcpy*, *strncpy*, *strlen*, *strchr*, *strrchr*, *strpbrk*, *strspn*, *strcspn*, *strtok* — string operations

## SYNOPSIS

```
#include <string.h>

char *strcat (s1, s2)
char *s1, *s2;

char *strncat (s1, s2, n)
char *s1, *s2;
int n;

int strcmp (s1, s2)
char *s1, *s2;

int strncmp (s1, s2, n)
char *s1, *s2;
int n;

char *strcpy (s1, s2)
char *s1, *s2;

char *strncpy (s1, s2, n)
char *s1, *s2;
int n;

int strlen (s)
char *s;

char *strchr (s, c)
char *s, c;

char *strrchr (s, c)
char *s, c;

char *strpbrk (s1, s2)
char *s1, *s2;

int strspn (s1, s2)
char *s1, *s2;

int strcspn (s1, s2)
char *s1, *s2;

char *strtok (s1, s2)
char *s1, *s2;
```

## DESCRIPTION

The arguments *s1*, *s2* and *s* point to strings (arrays of characters terminated by a null character). The functions *strcat*, *strncat*, *strcpy* and *strncpy* all alter *s1*. These functions do not check for overflow of the array pointed to by *s1*.

*Strcat* appends a copy of string *s2* to the end of string *s1*. *Strncat* appends at most *n* characters. Each returns a pointer to the null-terminated result.

*Strcmp* compares its arguments and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*. *Strncmp* makes the same comparison but looks at at most *n* characters.

*Strcpy* copies string *s2* to *s1*, stopping after the null character has been copied. *Strncpy* copies exactly *n* characters, truncating *s2* or adding null characters to *s1* if necessary. The result will not be null-terminated if the length of *s2* is *n* or more. Each function returns *s1*.

*Strlen* returns the number of characters in *s*, not including the terminating null character.

*Strchr* (*strchr*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or a NULL pointer if *c* does not occur in the string. The null character terminating a string is considered to be part of the string.

*Strpbrk* returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or a NULL pointer if no character from *s2* exists in *s1*.

*Strspn* (*strcspn*) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

*Strtok* considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and will have written a null character into *s1* immediately following the returned token. The function keeps track of its position in the string between separate calls, so that on subsequent calls (which must be made with the first argument a NULL pointer) will work through the string *s1* immediately following that token. In this way subsequent calls will work through the string *s1* until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, a NULL pointer is returned.

#### NOTE

For user convenience, all these functions are declared in the optional `<string.h>` header file.

#### BUGS

*Strcmp* and *strncmp* use native character comparison, which is signed on PDP-11s, unsigned on other machines.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

**NAME**

strtol, atol, atoi — convert string to integer

**SYNOPSIS**

**long strtol (str, ptr, base)**

**char \*str;**

**char \*\*ptr;**

**int base;**

**long atol (str)**

**char \*str;**

**int atoi (str)**

**char \*str;**

**DESCRIPTION**

*Strtol* returns as a long integer the value represented by the character string *str*. The string is scanned up to the first character inconsistent with the base. Leading “white-space” characters are ignored.

If the value of *ptr* is not (char \*\*)NULL, a pointer to the character terminating the scan is returned in *\*ptr*. If no integer can be formed, *\*ptr* is set to *str*, and zero is returned.

If *base* is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and “0x” or “0X” is ignored if *base* is 16.

If *base* is zero, the string itself determines the base thus: After an optional leading sign, a leading zero indicates octal conversion, and a leading “0x” or “0X” hexadecimal conversion. Otherwise, decimal conversion is used.

Truncation from long to int can, of course, take place upon assignment, or by an explicit cast.

*Atol(str)* is equivalent to *strtol(str, (char \*\*)NULL, 10)*.

*Atoi(str)* is equivalent to *(int) strtol(str, (char \*\*)NULL, 10)*.

**SEE ALSO**

atof(3C), scanf(3S).

**BUGS**

Overflow conditions are ignored.

**NAME**

swab — swap bytes

**SYNOPSIS**

```
void swab (from, to, nbytes)
char *from, *to;
int nbytes;
```

**DESCRIPTION**

*Swab* copies *nbytes* bytes pointed to by *from* to the array pointed to by *to*, exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP-11s and other machines. *Nbytes* should be even and non-negative. If *nbytes* is odd and positive *swab* uses *nbytes*−1 instead. If *nbytes* is negative *swab* does nothing.

**NAME**

system — issue a shell command from Fortran

**SYNOPSIS**

character\*N c

call system(c)

**DESCRIPTION**

*System* causes its character argument to be given to *sh*(1) as input, as if the string had been typed at a terminal. The current process waits until the shell has completed.

**SEE ALSO**

sh(1), exec(2), system(3S).

**NAME**

`system` — issue a shell command

**SYNOPSIS**

```
#include <stdio.h>
```

```
int system (string)
```

```
char *string;
```

**DESCRIPTION**

*System* causes the *string* to be given to *sh*(1) as input, as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

**FILES**

`/bin/sh`

**SEE ALSO**

`sh`(1), `exec`(2).

**DIAGNOSTICS**

*System* forks to create a child process that in turn `exec`'s `/bin/sh` in order to execute *string*. If the fork or `exec` fails, *system* returns `-1` and sets *errno*.

**NAME**

tan, dtan — Fortran tangent intrinsic function

**SYNOPSIS**

**real** r1, r2  
**double precision** dp1, dp2  
**r2 = tan(r1)**  
**dp2 = dtan(dp1)**  
**dp2 = tan(dp1)**

**DESCRIPTION**

*Tan* returns the real tangent of its real argument. *Dtan* returns the double-precision tangent of its double-precision argument. The generic *tan* function becomes *dtan* as required with a double-precision argument.

**SEE ALSO**

trig(3M).

**NAME**

`tanh`, `dtanh` — Fortran hyperbolic tangent intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
r2 = tanh(r1)
dp2 = dtanh(dp1)
dp2 = tanh(dp1)
```

**DESCRIPTION**

*Tanh* returns the real hyperbolic tangent of its real argument. *Dtanh* returns the double-precision hyperbolic tangent of its double precision argument. The generic form *tanh* may be used to return a double-precision value given a double-precision argument.

**SEE ALSO**

`sinh(3M)`.



**NAME**

tmpfile — create a temporary file

**SYNOPSIS**

```
#include <stdio.h>
```

```
FILE *tmpfile ()
```

**DESCRIPTION**

*Tmpfile* creates a temporary file and returns a corresponding FILE pointer. The file will automatically be deleted when the process using it terminates. The file is opened for update.

**SEE ALSO**

creat(2), unlink(2), fopen(3S), mktemp(3C), tmpnam(3S).

**NAME**

tmpnam, tmpnam — create a name for a temporary file

**SYNOPSIS**

```
#include <stdio.h>

char *tmpnam (s)
char *s;

char *tempnam (dir, pfx)
char *dir, *pfx;
```

**DESCRIPTION**

These functions generate file names that can safely be used for a temporary file.

*Tmpnam* always generates a file name using the path-name defined as **P\_tmpdir** in the *<stdio.h>* header file. If *s* is NULL, *tmpnam* leaves its result in an internal static area and returns a pointer to that area. The next call to *tmpnam* will destroy the contents of the area. If *s* is not NULL, it is assumed to be the address of an array of at least **L\_tmpnam** bytes, where **L\_tmpnam** is a constant defined in *<stdio.h>*; *tmpnam* places its result in that array and returns *s*.

*Tempnam* allows the user to control the choice of a directory. The argument *dir* points to the path-name of the directory in which the file is to be created. If *dir* is NULL or points to a string which is not a path-name for an appropriate directory, the path-name defined as **P\_tmpdir** in the *<stdio.h>* header file is used. If that path-name is not accessible, **/tmp** will be used as a last resort. This entire sequence can be up-staged by providing an environment variable **TMPDIR** in the user's environment, whose value is a path-name for the desired temporary-file directory.

Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the *pfx* argument for this. This argument may be NULL or point to a string of up to five characters to be used as the first few characters of the temporary-file name.

*Tempnam* uses *malloc(3C)* to get space for the constructed file name, and returns a pointer to this area. Thus, any pointer value returned from *tempnam* may serve as an argument to *free* (see *malloc(3C)*). If *tempnam* cannot return the expected result for any reason, i.e. *malloc* failed, or none of the above mentioned attempts to find an appropriate directory was successful, a NULL pointer will be returned.

**NOTES**

These functions generate a different file name each time they are called.

Files created using these functions and either *fopen* or *creat* are temporary only in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to use *unlink(2)* to remove the file when its use is ended.

**SEE ALSO**

*creat(2)*, *unlink(2)*, *fopen(3S)*, *malloc(3C)*, *mktemp(3C)*, *tmpfile(3S)*.

**BUGS**

If called more than 17,576 times in a single process, these functions will start recycling previously used names.

Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using these functions or *mktemp*, and the file names are chosen so as to render duplication by other means unlikely.

## NAME

sin, cos, tan, asin, acos, atan, atan2 — trigonometric functions

## SYNOPSIS

```
#include <math.h>
double sin (x)
double x;
double cos (x)
double x;
double tan (x)
double x;
double asin (x)
double x;
double acos (x)
double x;
double atan (x)
double x;
double atan2 (y, x)
double x, y;
```

## DESCRIPTION

*Sin*, *cos* and *tan* return respectively the sine, cosine and tangent of their argument, which is in radians.

*Asin* returns the arcsine of  $x$ , in the range  $-\pi/2$  to  $\pi/2$ .

*Acos* returns the arccosine of  $x$ , in the range 0 to  $\pi$ .

*Atan* returns the arctangent of  $x$ , in the range  $-\pi/2$  to  $\pi/2$ .

*Atan2* returns the arctangent of  $y/x$ , in the range  $-\pi$  to  $\pi$ , using the signs of both arguments to determine the quadrant of the return value.

## DIAGNOSTICS

*Sin*, *cos* and *tan* lose accuracy when their argument is far from zero. For arguments sufficiently large, these functions return 0 when there would otherwise be a complete loss of significance. In this case a message indicating TLOSS error is printed on the standard error output. For less extreme arguments, a PLOSS error is generated but no message is printed. In both cases, *errno* is set to ERANGE.

*Tan* returns HUGE for an argument which is near an odd multiple of  $\pi/2$  when the correct value would overflow, and sets *errno* to ERANGE.

Arguments of magnitude greater than 1.0 cause *asin* and *acos* to return 0 and to set *errno* to EDOM. In addition, a message indicating DOMAIN error is printed on the standard error output.

These error-handling procedures may be changed with the function *matherr*(3M).

## SEE ALSO

*matherr*(3M).

## NAME

*tsearch*, *tdelete*, *twalk* — manage binary search trees

## SYNOPSIS

```
#include <search.h>

char *tsearch ((char *) key, (char **) rootp, compar)
int (*compar)( );

char *tdelete ((char *) key, (char **) rootp, compar)
int (*compar)( );

void twalk ((char *) root, action)
void (*action)( );
```

## DESCRIPTION

*Tsearch* is a binary tree search routine generalized from Knuth (6.2.2) Algorithm T. It returns a pointer into a tree indicating where a datum may be found. If the datum does not occur, it is added at an appropriate point in the tree. *Key* points to the datum to be sought in the tree. *Rootp* points to a variable that points to the root of the tree. A NULL pointer value for the variable denotes an empty tree; in this case, the variable will be set to point to the datum at the root of the new tree. *Compar* is the name of the comparison function. It is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero according as the first argument is to be considered less than, equal to, or greater than the second.

*Tdelete* deletes a node from a binary search tree. It is generalized from Knuth (6.2.2) algorithm D. The arguments are the same as for *tsearch*. The variable pointed to by *rootp* will be changed if the deleted node was the root of the tree. *Tdelete* returns a pointer to the parent of the deleted node, or a NULL pointer if the node is not found.

*Twalk* traverses a binary search tree. *Root* is the root of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) *Action* is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The second argument is a value from an enumeration data type `typedef enum { preorder, postorder, endorder, leaf } VISIT`; (defined in the `<search.h>` header file), depending on whether this is the first, second or third time that the node has been visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument is the level of the node in the tree, with the root being level zero.

## NOTES

The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

Warning: the *root* argument to *twalk* is one level of indirection less than the *rootp* arguments to *tsearch* and *tdelete*.

## DIAGNOSTICS

A NULL pointer is returned by *tsearch* if there is not enough space available to create a new node.

A NULL pointer is returned by *tsearch* and *tdelete* if *rootp* is NULL on entry.

**SEE ALSO**

bsearch(3C), hsearch(3C), lsearch(3C).

**BUGS**

Awful things can happen if the calling function alters the pointer to the root.

**NAME**

`ttyname`, `isatty` — find name of a terminal

**SYNOPSIS**

```
char *ttyname (fildes)
int fildes;

int isatty (fildes)
int fildes;
```

**DESCRIPTION**

*Ttyname* returns a pointer to a string containing the null-terminated path name of the terminal device associated with file descriptor *fildes*.

*Isatty* returns 1 if *fildes* is associated with a terminal device, 0 otherwise.

**FILES**

`/dev/*`

**DIAGNOSTICS**

*Ttyname* returns a NULL pointer if *fildes* does not describe a terminal device in directory `/dev`.

**BUGS**

The return value points to static data whose content is overwritten by each call.

**NAME**

ttyslot — find the slot in the utmp file of the current user

**SYNOPSIS**

int ttyslot ( )

**DESCRIPTION**

*Ttyslot* returns the index of the current user's entry in the */etc/utmp* file. This is accomplished by actually scanning the file */etc/inittab* for the name of the terminal associated with the standard input, the standard output, or the error output (0, 1 or 2).

**FILES**

*/etc/inittab*  
*/etc/utmp*

**SEE ALSO**

getut(3C), ttyname(3C).

**DIAGNOSTICS**

A value of 0 is returned if an error was encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device.

**NAME**

`ungetc` — push character back into input stream

**SYNOPSIS**

```
#include <stdio.h>
int ungetc (c, stream)
char c;
FILE *stream;
```

**DESCRIPTION**

*Ungetc* inserts the character *c* into the buffer associated with an input *stream*. That character, *c*, will be returned by the next *getc* call on that *stream*. *Ungetc* returns *c*, and leaves the file *stream* unchanged.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered.

If *c* equals EOF, *ungetc* does nothing to the buffer and returns EOF.

*Fseek*(3S) erases all memory of inserted characters.

**SEE ALSO**

*fseek*(3S), *getc*(3S), *setbuf*(3S).

**DIAGNOSTICS**

In order that *ungetc* perform correctly, a read statement must have been performed prior to the call of the *ungetc* function. *Ungetc* returns EOF if it can't insert the character. In the case that *stream* is *stdin*, *ungetc* will allow exactly one character to be pushed back onto the buffer without a previous read statement.



## NAME

x25alnk, x25ilnk — attach or install a BX.25 link

## SYNOPSIS

```
#include <x25lib.h>

int x25alnk (linkid, devname, lineno, modname,
int linkid, lineno;
char *devname, *modname;
unsigned flags;

int x25ilnk (linkid, pktsize, flags)
int linkid, pktsize;
unsigned flags;
```

## DESCRIPTION

*X25alnk* is used to attach a BX.25 logical link specified by *linkid* to a level 2 device whose name is *devname* by making the necessary connections between data structures.

*Linkid* is the identifier for the link data structure to be used in the operating system. This identifier can be thought of as the connector between *x25ipvc* calls and the *x25alnk* call for the physical link on which the channels are multiplexed. An example of a link identifier is 1.

*Devname* is the name of the physical device running the interpreter and script for this link, e.g., */dev/kmc0*.

*Lineno* is the physical line number (range 0-7) for a logical link on a physical unit, e.g., 4.

*Modname* is the name of the synchronous modem control device. If the LNKMOD flag is specified, the standard modem control functions performed for the line are to raise data terminal ready and request to send. An example of *modname* is */dev/kdm0*.

*Flags* specifies the options for the attach call, e.g., LNKBACK requests *devname* as a backup device. The permissible *flags* bit settings for attach are:

```
LNKMOD  modname specified.
LNKBACK  attach a backup rather a primary device.
```

*X25ilnk* is used to initialize a link; more precisely, to start the level 2 protocol in the associated device and to start the level 3 protocol in the UNIX driver for the link specified by *linkid*.

*Pktsize* is the packet size to be used for level 3 data packets. *Pktsize* must be a number that is a power of 2 between 16 and 1024 inclusive. The default packet size is 128. The LNKPKT flag must be raised to set a non-default size.

*Flags* specifies the options for the initialization call, e.g., LNKISB requests the B address. The permissible *flags* bit settings for initialization are:

```
LNKPKT  packet size specified
LNKISB  tell interpreter line is an X.25 B address; default is A.
LNKBACK  initialize the backup device.
LNKFAST  the device speed is greater than 9.6 KB.
```

## SEE ALSO

ioctl(2), open(2), stat(2), perror(3C), x25clnk(3C), x25hlnk(3C), x25pvc(1M), nc(7), vpm(7), x25(7) in the *UNIX System Administrator's Manual*.

*Operations Systems Network Protocol Specification: BX.25 Issue 2*, Bell Laboratories.

**DIAGNOSTICS**

ELNKPKT packet size specified is illegal.  
ELNKNCO network control device *open* failed; check *errno*.  
ELNKNCI network control device *ioctl* failed; check *errno*.  
ELNKDS *stat* of physical device failed; check *errno*.  
ELNKDNC file associated with device name not a character special device.  
ELNKMCO modem control device *open* failed; check *errno*.  
ELNKMCI modem control device *ioctl* failed; check *errno*.  
ELNKLNO device line number illegal.

## X25CLNK(3C)

### NAME

x25clnk — change over a BX.25 link

### SYNOPSIS

```
#include <x25lib.h>
int x25clnk (linkid)
int linkid;
```

### DESCRIPTION

*X25clnk* is used to change over from the primary to the backup level 2 device associated with link *linkid*. *Linkid* is the identifier for the link data structure which is used in the operating system. This identifier was set up by the *x25alnk* subroutine call.

### SEE ALSO

*ioctl(2)*, *open(2)*, *stat(2)*, *perror(3C)*, *x25alnk(3C)*, *x25hlnk(3C)*, *x25pvc(1M)*, *nc(7)*, *vpm(7)*, *x25(7)* in the *UNIX System Administrator's Manual*.  
*Operations Systems Network Protocol Specification: BX.25 Issue 2*, Bell Laboratories.

### DIAGNOSTICS

ELNKNCO network control device *open* failed; check *errno*.  
ELNKNCI network control device *ioctl* failed; check *errno*.  
ELNKDS *stat* of physical device failed; check *errno*.  
ELNKDNC file associated with device name not a character special device.

**NAME**

x25hlnk, x25dlnk — halt or detach a BX.25 link

**SYNOPSIS**

```
#include <x25lib.h>
int x25hlnk (linkid, flags)
int linkid;
unsigned flags;
int x25dlnk (linkid, flags)
int linkid;
unsigned flags;
```

**DESCRIPTION**

*X25hlnk* is used to halt a link; more precisely, to stop the level 2 protocol in the associated device and to stop the level 3 protocol in the UNIX driver for the link specified by *linkid*. If a backup device has been attached and started, the level 2 protocol on the backup will also be stopped.

*X25dlnk* is used to detach a BX.25 logical link specified by *linkid*. This removes the logical connections which were made by *x25alnk*.

*Linkid* is the identifier for the link data structure which is used in the operating system. This identifier was set up by the *x25alnk* subroutine call.

*Flags* specifies the options for the halt or detach call.

The permissible *flags* bit settings for halt is:

LNKBACK halt only the level 2 protocol on the backup device. The level 3 protocol must not be running on this backup device.

The permissible *flags* bit settings for detach is:

LNKBACK detach a backup rather than a primary device.

**SEE ALSO**

ioctl(2), open(2), stat(2), perror(3C), x25alnk(3C), x25clnk(3C), x25pvc(1M), nc(7), vpm(7), x25(7) in the *UNIX System Administrator's Manual*.

*Operations Systems Network Protocol Specification: BX.25 Issue 2*, Bell Laboratories.

**DIAGNOSTICS**

ELNKNCO network control device *open* failed; check *errno*.

ELNKNCI network control device *ioctl* failed; check *errno*.

ELNKDS *stat* of physical device failed; check *errno*.

ELNKDNC file associated with device name not a character special device.

## NAME

x25ipvc, x25rpvc — install or remove a PVC on a link

## SYNOPSIS

```
#include <x25lib.h>
int x25ipvc (slotname, chno, linkid, flags)
char *slotname;
int chno, linkid;
unsigned flags;
int x25rpvc (slotname)
char *slotname;
```

## DESCRIPTION

*X25ipvc* may be used to install a BX.25 Permanent Virtual Circuit (PVC) on a specified BX.25 interface (*link*). If *slotname* is currently connected (but removable) this connection is removed and the new connection is made to the logical channel *chno* on the link specified by *linkid*.

*Slotname* is a path name that specifies a BX.25 minor device (slot), e.g., */dev/x25s12*.

*Chno* is the BX.25 level 3 logical channel number associated with the PVC, e.g., 3. *chno* must be in the range 1 to 4095 and must not be currently in use for any other BX.25 minor device associated with that link.

*Linkid* is the identifier for the link data structure to be used in the operating system. This identifier can be thought of as the connector between *x25ipvc* calls and the *x25alnk* call for the physical link on which the channels are multiplexed. An example of a link identifier is 1.

*Flags* contains settings for specifying PVC install options; permissible PVC *flags* bit settings are:

```
PVCSESS  Session connect/disconnect packets used.
PVCREST  RESET in-order/out-of-order responded to.
PVCNONE  No establishment protocol used.
```

*X25rpvc* is used to remove the association between BX.25 minor device *slotname* and the link and channel to which it is currently connected. The command will fail if the slot is open, if packets are waiting to be transmitted, or if there are unacknowledged packets outstanding.

## SEE ALSO

ioctl(2), open(2), stat(2), perror(3C).  
 nc(7), vpm(7), x25(7) in the *UNIX System Administrator's Manual*.  
*Operations Systems Network Protocol Specification: BX.25 Issue 2*, Bell Laboratories.

## DIAGNOSTICS

```
EPVCNP   no (or multiple) setup protocol specified (one of PVCSESS,
          PVCREST, or PVCNONE must be in flags argument).
EPVCNCO  network control device open failed; check errno.
EPVCNCI  network control device ioctl failed; check errno.
EPVCSS   stat of slot (PVC) name failed; check errno.
EPVCSNC  file associated with slotname not a character special device.
```

**NAME**

intro — introduction to file formats

**DESCRIPTION**

This section outlines the formats of various files. The C **struct** declarations for the file formats are given where applicable. Usually, these structures can be found in the directories `/usr/include` or `/usr/include/sys`.

References of the type *name*(1M) refer to entries found in Section 1 of the *UNIX System Administrator's Manual*.

**NAME**

a.out — common assembler and link editor output

**DESCRIPTION**

The file name **a.out** is the output file from the assembler *as(1)* and the link editor *ld(1)*. Both programs will make *a.out* executable if there were no errors in assembling or linking and no unresolved external references.

A common object file consists of a file header, a UNIX header, a table of section headers, relocation information, (optional) line numbers, and a symbol table. The order is given below.

```

File header.
UNIX header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
...
Section n line numbers.
Symbol table.
```

The last three sections (relocation, line numbers and symbol table) may be missing if the program was linked with the *-s* option of *ld(1)* or if the symbol table and relocation bits were removed by *strip(1)*. Also note that if there were no unresolved external references after linking, the relocation information will be absent.

The sizes of each segment (contained in the header, discussed below) are in bytes and are even.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. The text segment begins at location 0 in the core image; the header is not loaded. If the magic number (the first field in the UNIX header) is 407 (octal), it indicates that the text segment is not to be write-protected or shared, so the data segment will be contiguous with the text segment. If the magic number is 410 (octal), the data segment begins at the next segment boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same **a.out** file, they will share a single text segment.

On the 3B20S, the stack begins at the end of the text and data sections and grows towards higher addresses. On the VAX, the stack begins at the end of memory and grows towards lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the *brk(2)* system call.

The value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, the storage class of the symbol-table entry for that word will be marked as an "external symbol", and the

section number will be set to 0. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

### File Header

The format of the `filehdr` header is

```
struct filehdr
{
    unsigned short f_magic; /* magic number */
    unsigned short f_nscns; /* number of sections */
    long          f_timdat; /* time and date stamp */
    long          f_symptr; /* file ptr to symtab */
    long          f_nsyms; /* # symtab entries */
    unsigned short f_opthdr; /* sizeof(opt hdr) */
    unsigned short f_flags; /* flags */
};
```

### UNIX Header

The format of the UNIX header is

```
typedef struct aouthdr {
    short  magic; /* magic number */
    short  vstamp; /* version stamp */
    long   tsize; /* text size in bytes, padded */
    long   dsize; /* initialized data (.data) */
    long   bsize; /* uninitialized data (.bss) */
    long   entry; /* entry point */
    long   text_start; /* base of text used for this file */
    long   data_start; /* base of data used for this file */
} AOUTHDR;
```

### Section Header

The format of the section header is

```
struct scnhdr
{
    char          s_name[SYMNMLEN]; /* section name */
    long          s_paddr; /* physical address */
    long          s_vaddr; /* virtual address */
    long          s_size; /* section size */
    long          s_scnptr; /* file ptr to raw data */
    long          s_relptr; /* file ptr to relocation */
    long          s_innoptr; /* file ptr to line numbers */
    unsigned short s_nreloc; /* # reloc entries */
    unsigned short s_nlnno; /* # line number entries */
    long          s_flags; /* flags */
};
```



**Relocation**

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format:

```

struct reloc
{
    long    r_vaddr;    /* (virtual) address of reference */
    long    r_symndx;   /* index into symbol table */
    short   r_type;    /* relocation type */
};

```

The start of the relocation information is *relptr* from the Section Header. If there is no relocation information, *relptr* is 0.

**Symbol Table**

The format of the symbol table header is

```

#define SYMNMLEN  8
#define FILNMLEN  14
#define SYMESZ    18      /* the size of a SYMENT */

struct syment
{
    char          n_name[SYMNMLEN]; /* name of :  bo
    unsigned long n_value;          /* value of symbol */
    short        n_scnum; /* section number */
    unsigned short n_type; /* type and derived type */
    char         n_sclass; /* storage class */
    char         n_numaux; /* number of aux entries */
};

```

Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows:

```

union auxent {
    struct {
        long    x_tagndx;
        union {
            struct {
                unsigned short x_lnno;
                unsigned short x_size;
            } x_lnsz;
            long    x_fsize;
        } x_misc;
        union {
            struct {
                long    x_lnnoptr;
                long    x_endndx;
            } x_fcn;
            struct {
                unsigned short x_dimen[DIMNUM];
            } x_ary;
        } x_fcary;
        unsigned short x_tvndx;
    } x_sym;

    struct {
        char    x_fname[FILNMLEN];
    } x_file;

    struct {
        long    x_scrlen;
        unsigned short x_nreloc;
        unsigned short x_nlinno;
    } x_scn;

    struct {
        long    x_tvfill;
        unsigned short x_tvlen;
        unsigned short x_tvran[2];
    } x_tv;
};

```

Indexes of symbol table entries begin at *zero*. The start of the symbol table is *symptr* (from the file header) bytes from the beginning of the file. If the symbol table is stripped, *symptr* is 0.

**SEE ALSO**

as(1), cc(1), ld(1), filehdr(4), ldfcn(4), linenum(4), reloc(4), scnhdr(4), syms(4).

**NAME**

a.out — PDP-11 assembler and link editor output

**DESCRIPTION**

**A.out** is the output file of the assembler *as*(1) and the link editor *ld*(1). Both programs will make **a.out** executable if there were no errors in assembling or linking and no unresolved external references.

This file has four sections: a header, the program text and data segments, relocation information, and a symbol table (in that order). The last two sections may be missing if the program was linked with the *-s* option of *ld*(1) or if the symbol table and relocation bits were removed by *strip*(1). Also note that if there were no unresolved external references after linking, the relocation information will be removed.

The sizes of each segment (contained in the header, discussed below) are in bytes and are even. The size of the header is not included in any of the other sizes.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. The text segment begins at location 0 in the core image; the header is not loaded. If the magic number (the first field in the header) is 407 (octal), it indicates that the text segment is not to be write-protected or shared, so the data segment will be contiguous with the text segment. If the magic number is 410 (octal), the data segment begins at the first 0 mod 8K byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same **a.out** file, they will share a single text segment. If the magic number is 411 (octal) the text segment is again pure (write-protected and shared) and, moreover, the instruction and data spaces are separated; the text and data segment both begin at location 0. See the *PDP-11/70 Processor Handbook* for restrictions that apply to this situation.

The stack will occupy the highest possible locations in the core image: from 177776 (octal) on the PDP-11 and growing downwards. The stack is automatically extended as required. The data segment is only extended as requested by the *brk*(2) system call.

The start of the text segment in the **a.out** file is *hsize*; the start of the data segment is *hsize+S<sub>t</sub>* (the size of the text), where *hsize* is 20 (octal) on the PDP-11.

The value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation information (discussed below) for that word, then the value of the word as stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

**Header—PDP-11**

The format of the **a.out** header for the PDP-11 is as follows:

```

struct  exec      {
    short         a_magic; /* magic number */
    unsigned      a_text; /* size of text segment */
    unsigned      a_data; /* size of data segment */
    unsigned      a_bss; /* size of bss segment */
    unsigned      a_syms; /* size of symbol table */
    unsigned      a_entry; /* entry point of program */
    char          a_unused;
    char          a_hitext; /* hi bits for large text spaces */
    char          a_flag; /* set if relocation info stripped */
    char          a_stamp; /* version stamp */
};

```

**Relocation—PDP-11**

If relocation information is present, it amounts to two bytes per relocatable datum. There is no relocation information if the “suppress relocation” flag (*a\_flag*) in the header is on.

The format of the relocation data is:

```

struct  r_info {
    int    r_symbolnum:11,
          r_segment:3,
          r_pcrel:1;
};

```

The *r\_pcrel* field indicates, if *on*, that the reference is relative to the program counter (pc) register (e.g., *clr x*); if *off*, that the reference is to the actual symbol (e.g., *clr \*\$x*).

The *r\_segment* field indicates the segment referred to by the text or data word associated with the relocation word:

```

00 indicates the reference is absolute;
02 indicates the reference is to the text segment;
04 indicates the reference is to initialized data;
06 indicates the reference is to bss (uninitialized data);
10 indicates the reference is to an undefined external symbol.

```

The field *r\_symbolnum* contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered 0, the second 1, etc.

The start of the relocation information on the PDP-11 is:

$$hsize + a\_text + a\_data$$
**Symbol Table—PDP-11**

The symbol table on the PDP-11 consists of entries of the form:

```

struct  nlist {
    char    n_name[8];
    int     n_type;
    unsigned n_value;
};

```

The *n\_name* field contains the ASCII name of the symbol, null-padded. The *n\_type* field indicates the type of the symbol; the following values are possible:

00	undefined symbol
01	absolute symbol
02	text segment symbol
03	data segment symbol
04	bss segment symbol
37	file name symbol (produced by <i>ld(1)</i> )
40	undefined external symbol
41	absolute external symbol
42	text segment external symbol
43	data segment external symbol
44	bss segment external symbol

The start of the symbol table on the PDP-11 is:

$hsize + 2(a\_text + a\_data)$

if relocation information is present, and

$hsize + a\_text + a\_data$

if it is not.

If a symbol's type on the PDP-11 is *undefined external* and the value field is non-zero, the symbol is interpreted by the link editor *ld(1)* as the name of a common region whose size is indicated by the value of the symbol.

**SEE ALSO**

*as(1)*, *ld(1)*, *nm(1)*, *strip(1)*.

## NAME

acct — per-process accounting file format

## SYNOPSIS

```
#include <sys/acct.h>
```

## DESCRIPTION

Files produced as a result of calling *acct(2)* have records in the form defined by *<sys/acct.h>*, whose contents are:

```
typedef ushort comp_t; /* "floating point" */
                        /* 13-bit fraction, 3-bit exponent */

struct acct
{
    char    ac_flag;      /* Accounting flag */
    char    ac_stat;     /* Exit status */
    ushort  ac_uid;
    ushort  ac_gid;
    dev_t   ac_tty;
    time_t  ac_btime;    /* Beginning time */
    comp_t  ac_untime;   /* acctng user time in clock ticks */
    comp_t  ac_stime;    /* acctng system time in clock ticks */
    comp_t  ac_etime;    /* acctng elapsed time in clock ticks */
    comp_t  ac_mem;      /* memory usage in clicks */
    comp_t  ac_io;       /* chars trnsfrd by read/write */
    comp_t  ac_rw;       /* number of block reads/writes */
    char    ac_comm[8];  /* command name */
};

extern struct acct  acctbuf;
extern struct inode *acctp; /* inode of accounting file */

#define AFORK 01 /* has executed fork, but no exec */
#define ASU   02 /* used super-user privileges */
#define ACCTF 0300 /* record type: 00 = acct */
```

In *ac\_flag*, the AFORK flag is turned on by each *fork(2)* and turned off by an *exec(2)*. The *ac\_comm* field is inherited from the parent process and is reset by any *exec*. Each time the system charges the process with a clock tick, it also adds to *ac\_mem* the current process size, computed as follows:

$$(\text{data size}) + (\text{text size}) / (\text{number of in-core processes using text})$$

The value of  $ac\_mem / (ac\_stime + ac\_untime)$  can be viewed as an approximation to the mean process size, as modified by text-sharing.

The structure `tacct.h`, which resides with the source files of the accounting commands, represents the total accounting format used by the various accounting commands:

```

/*
 * total accounting (for acct period), also for day
 */

struct tacct {
    uid_t      ta_uid;      /* userid */
    char       ta_name[8]; /* login name */
    float      ta_cpu[2];  /* cum. cpu time, p/np (mins) */
    float      ta_kcore[2]; /* cum kcore-minutes, p/np */
    float      ta_con[2];  /* cum. connect time, p/np, mins */
    float      ta_du;      /* cum. disk usage */
    long       ta_pc;      /* count of processes */
    unsigned short ta_sc;  /* count of login sessions */
    unsigned short ta_dc;  /* count of disk samples */
    unsigned short ta_fee; /* fee for special services */
};

```

#### SEE ALSO

`acct(1M)`, `acctcom(1)`, `acct(2)`.

#### BUGS

The `ac_mem` value for a short-lived command gives little information about the actual size of the command, because `ac_mem` may be incremented while a different command (e.g., the shell) is being executed by the process.

## NAME

ar — common archive file format

## DESCRIPTION

The archive command *ar* is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor *ld*(1).

Each archive begins with an archive file header which is made up of the following components:

```
#define ARMAG      "<ar>"
#define SARMAG     4

struct ar_hdr {
    char    ar_magic[SARMAG];    /* magic number */
    char    ar_name[16];        /* archive name */
    char    ar_date[4];         /* date of last archive modification */
    char    ar_syms[4];         /* number of ar_sym entries */
};
```

Each archive which contains common object files (see *a.out*(4)) includes an archive symbol table. This symbol table is used by the link editor *ld*(1) to determine which archive members must be loaded during the link edit process. The archive file header described above is followed by a number of symbol table entries. The number of symbol table entries is indicated in the *ar\_syms* variable. Each symbol table entry has the following format:

```
struct ar_sym {
    char    sym_name[8];        /* symbol name, recognized by ld */
    char    sym_ptr[4];         /* archive position of symbol */
};
```

The archive symbol table is automatically created and/or updated by the *ar*(1) command.

Following the archive header and symbol table are the archive file members. Each file member is preceded by a file member header which is of the following format:

```
struct arf_hdr {
    char    arf_name[16];      /* file member name */
    char    arf_date[4];       /* file member date */
    char    arf_uid[4];        /* file member user identification */
    char    arf_gid[4];        /* file member group identification */
    char    arf_mode[4];       /* file member mode */
    char    arf_size[4];       /* file member size */
};
```

All information in the archive header, symbol table and file member headers is stored in a machine independent fashion. All character data is automatically portable. The numeric information contained in the headers is also stored in a machine independent fashion. All numeric data is stored as four bytes and is accessed by the special archive I/O functions described in *spu1*(3X) functions of the *libld.a* library. Common format archives can be moved from system to system as long as the portable archive command



*ar(1)* is used. Conversion tools such as *arcv(1)* and *convert(1)* exist to aid in the transportation of non-common format archives to this format.

Each archive file member begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

**SEE ALSO**

*ar(1)*, *arcv(1)*, *convert(1)*, *ld(1)*, *sputl(3X)*.

**BUGS**

The common archive structure is not compatible between the PDP-11 and the IBM-370, due to the different file formats. See *arcv(1)* and *convert(1)* to convert between machines.

*Strip(1)* will remove all archive symbol entries from the header. The archive symbol entries must be restored via the *s* option of the *ar(1)* command before the archive can be used with the link editor *ld(1)*.

**NAME**

ar — archive file format

**DESCRIPTION**

The archive command *ar* is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor *ld*(1).

A file produced by *ar* has a magic number at the start, followed by the constituent files, each preceded by a file header. The magic number is 0177545(octal) (it was chosen to be unlikely to occur anywhere else). The header of each file is 26 bytes long:

```
#define ARMAG 0177545
struct ar_hdr {
    char ar_name[14];
    long ar_date;
    char ar_uid;
    char ar_gid;
    int ar_mode;
    long ar_size;
};
```

Each file begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

**SEE ALSO**

ar(1), ld(1).

**NAME**

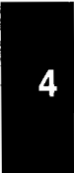
checklist — list of file systems processed by fsck

**DESCRIPTION**

*Checklist* resides in directory */etc* and contains a list of at most 15 *special file* names. Each *special file* name is contained on a separate line and corresponds to a file system. Each file system will then be automatically processed by the *fsck(1M)* command.

**SEE ALSO**

*fsck(1M)*.



**NAME**

core — format of core image file

**DESCRIPTION**

UNIX writes out a core image of a terminated process when any of various errors occur. See *signal(2)* for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called **core** and is written in the process's working directory (provided it can be; normal access controls apply). A process with an effective user ID different from the real user ID will not produce a core image.

The first section of the core image is a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter *usize*, which is defined in */usr/include/sys/param.h*. The remainder represents the actual contents of the user's core area when the core image was written. If the text segment is read-only and shared, or separated from data space, it is not dumped.

The format of the information in the first section is described by the *user* structure of the system, defined in */usr/include/sys/user.h*. The important stuff not detailed therein is the locations of the registers, which are outlined in */usr/include/sys/reg.h*.

**SEE ALSO**

crash(1M), sdb(1), setuid(2), signal(2).

**NAME**

cpio — format of cpio archive

**DESCRIPTION**

The *header* structure, when the `-c` option of *cpio*(1) is not used, is:

```

struct {
    short    h_magic,
            h_dev;
    ushort   h_ino,
            h_mode,
            h_uid,
            h_gid;
    short    h_nlink,
            h_rdev,
            h_mtime[2],
            h_namesize,
            h_filesize[2];
    char     h_name[h_namesize rounded to word];
} Hdr;

```

When the `-c` option is used, the *header* information is described by:

```

sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
        &Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
        &Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
        &Longtime, &Hdr.h_namesize,&Longfile,Hdr.h_name);

```

*Longtime* and *Longfile* are equivalent to *Hdr.h\_mtime* and *Hdr.h\_filesize*, respectively. The contents of each file are recorded in an element of the array of varying length structures, *archive*, together with other items describing the file. Every instance of *h\_magic* contains the constant 070707 (octal). The items *h\_dev* through *h\_mtime* have meanings explained in *stat*(2). The length of the null-terminated path name *h\_name*, including the null byte, is given by *h\_namesize*.

The last record of the *archive* always contains the name TRAILER!!!. Special files, directories, and the trailer are recorded with *h\_filesize* equal to zero.

**SEE ALSO**

*cpio*(1), *find*(1), *stat*(2).

**NAME**

dir — format of directories

**SYNOPSIS****#include** <sys/dir.h>**DESCRIPTION**

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry (see *fs(4)*). The structure of a directory entry as given in the include file is:

```

#ifdef DIRSIZ
#define DIRSIZ    14
#endif
struct  direct
{
        ino_t  d_ino;
        char  d_name[DIRSIZ];
};

```

By convention, the first two entries in each directory are for . and ... The first is an entry for the directory itself. The second is for the parent directory. The meaning of .. is modified for the root directory of the master file system; there is no parent, so .. has the same meaning as ..

**SEE ALSO**

fs(4).

**NAME**

errfile — error-log file format

**DESCRIPTION**

When hardware errors are detected by the system, an error record is generated and passed to the error-logging daemon for recording in the error log for later analysis. The default error log is `/usr/adm/errfile`.

The format of an error record depends on the type of error that was encountered. Every record, however, has a header with the following format:

```
struct errhdr {
    short      e_type;      /* record type */
    short      e_len;       /* bytes in record (inc hdr) */
    time_t     e_time;     /* time of day */
};
```

The permissible record types are as follows:

```
#define E_GOTS      010      /* start for UNIX 3.0 */
#define E_GORT      011      /* start for UNIX/RT */
#define E_STOP      012      /* stop */
#define E_TCHG      013      /* time change */
#define E_CCHG      014      /* configuration change */
#define E_BLK       020      /* block device error */
#define E_STRAY     030      /* stray interrupt */
#define E_PRTY     031      /* memory parity */
#define E_PIO       041      /* 3B-20 programmed I/O */
#define E_IOP       042      /* 3B-20 I/O processor */
```

Some records in the error file are of an administrative nature. These include the startup record that is entered into the file when logging is activated, the stop record that is written if the daemon is terminated “gracefully”, and the time-change record that is used to account for changes in the system’s time-of-day. These records have the following formats:

```
struct estart {
    short      e_cpu;       /* CPU type */
    struct utsname e_name;  /* system names */
#ifdef u3b
    short      e_mmr3;     /* contents mem mgmt reg 3 */
    long       e_syssize;  /* 11/70 system memory size */
    short      e_bconf;    /* block dev configuration */
#endif
#ifdef u3b
    int        e_mmcnt;    /* kbytes per array */
#endif
};

#define eend errhdr /* record header */

struct etimchg {
    time_t     e_nptime;   /* new time */
};
```

Stray interrupts cause a record with the following format to be logged:

```
struct estray {
#ifdef u3b
    uint          e_saddr;    /* stray loc or device addr */
#else
    physadr      e_saddr;    /* stray loc or device addr */
    short        e_sbacty;   /* active block devices */
#endif
};
```

Memory subsystem error on 3B-20 and 11/70 processors cause the following record to be generated:

```
struct eparity {
#ifdef u3b
    int          e_parreg[3]; /* 3B memory registers */
#else
    short        e_parreg[4]; /* memory subsys registers */
#endif
};
```

Memory subsystem errors on VAX-11/780 processors cause the following record to be generated:

```
struct ememory {
    int          e_sbier;
    int          e_memcad;
};
```

Error records for block devices have the following format:

```
struct eblock {
#ifdef u3b
    ushort       e_num;      /* device number */
    struct iostat {
        long      io_ops;    /* number read/writes */
        long      io_misc;   /* number "other" operations */
        ushort    io_unlog;  /* number unlogged errors */
    }
    e_stats;
    short        e_bflags;   /* read/write, error, etc */
    daddr_t      e_bnum;     /* logical block number */
    uint         e_bytes;    /* number bytes to transfer */
    union ptbl {
        int page[64];       /* page table entries */
        union ptbl *pnext;
    }
    struct ptbl  e_ptbl;     /* page table for transfer */
    uint         e_voff;     /* offset into page table */
    uint         e_stat1;    /* status word 1 */
    uint         e_stat2;    /* status word 2 */
#endif
};
```



```

#ifdef u3b
    dev_t      e_dev;      /* "true" major + minor dev no */
    physadr    e_regloc;   /* controller address */
    short      e_bacty;    /* other block I/O activity */
    struct iostat {
        long    io_ops;    /* number read/writes */
        long    io_misc;   /* number "other" operations */
        ushort  io_unlog;  /* number unlogged errors */
    }
    e_stats;
    short      e_bflags;   /* read/write, error, etc */
    short      e_cyloff;   /* logical dev start cyl */
    daddr_t    e_bnum;    /* logical block number */
    ushort     e_bytes;    /* number bytes to transfer */
    paddr_t    e_memadd;  /* buffer memory address */
    ushort     e_rtry;    /* number retries */
    short      e_nreg;    /* number device registers */
#endif
#ifdef vax
    struct mba_regs {
        long mba_csr;
        long mba_cr;
        long mba_sr;
        long mba_var;
        long mba_vcr;
    } e_mba;
#endif
};

```

The following values are used in the *e\_bflags* word:

```

#define E_WRITE  0      /* write operation */
#define E_READ   1      /* read operation */
#define E_NOIO   02     /* no I/O pending */
#define E_PHYS   04     /* physical I/O */
#define E_MAP    010    /* Unibus map in use */
#define E_ERROR  020    /* I/O failed */

```

The following error records are for the 3B-20 only:

```

struct epio {
    char      e_chan;     /* programmed I/O (pio) error */
    char      e_dev;     /* which channel */
    uint      e_chstat;  /* which dev on channel */
    uint      e_cmd;     /* channel status */
    uint      e_cmd;     /* pio command */
}

struct eiop {
    char      e_unit;    /* I/O processor (iop) error */
    uint      e_word0;  /* unit number */
    uint      e_word0;  /* iop report word */
    uint      e_word1;  /* iop report word */
}

```

The "true" major device numbers that identify the failing device are as follows:

<i>Digital Equipment</i>		<i>Western Electric</i>	
#define RK0	0	#define DFC0	0
#define RP0	1	#define IOPO	1
#define RF0	2	#define MT0	2
#define TM0	3		
#define TC0	4		
#define HP0	5		
#define HT0	6		
#define HS0	7		
#define RL0	8		
#define HP1	9		
#define HP2	10		
#define HP3	11		

**SEE ALSO**

errdemon(1M).

## NAME

filehdr — file header for common object files

## SYNOPSIS

```
#include <filehdr.h>
```

## DESCRIPTION

Every common object file begins with a 20-byte header. The following C struct declaration is used:

```
struct filehdr
{
    unsigned short f_magic ; /* magic number */
    unsigned short f_nscns ; /* number of sections */
    long          f_timdat ; /* time & date stamp */
    long          f_symptr ; /* file ptr to symtab */
    long          f_nsyms ; /* # symtab entries */
    unsigned short f_opthdr ; /* sizeof(opt hdr) */
    unsigned short f_flags ; /* flags */
};
```

*F\_symptr* is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in *fseek*(3S) to position an I/O stream to the symbol table. The UNIX optional header is always 36 bytes. The valid magic numbers are given below:

```
#define N3BMAGIC 0550 /* 3B20S */
#define NTVMAGIC 0551 /* 3B20S */

#define VAXWRMAGIC 0570 /* VAX writable text segments */
#define VAXROMAGIC 0575 /* VAX readonly sharable text segments */
```

The value in *f\_timdat* is obtained from the *time*(2) system call. Flag bits currently defined are:

```
#define F_RELFLG 00001 /* relocation entries stripped */
#define F_EXEC 00002 /* file is executable */
#define F_LNNO 00004 /* line numbers stripped */
#define F_LSYMS 00010 /* local symbols stripped */
#define F_MINMAL 00020 /* minimal object file */
#define F_UPDATE 00040 /* update file, ogen produced */
#define F_SWABD 00100 /* file is "pre-swabbed" */
#define F_AR16WR 00200 /* 16 bit DEC host */
#define F_AR32WR 00400 /* 32 bit DEC host */
#define F_AR32W 01000 /* non-DEC host */
#define F_PATCH 02000 /* "patch" list in opt hdr */
```

## SEE ALSO

*time*(2), *fseek*(3S), *a.out*(4).

## NAME

file system — format of system volume

## SYNOPSIS

```
#include <sys/filsys.h>
#include <sys/types.h>
#include <sys/param.h>
```

## DESCRIPTION

Every file system storage volume has a common format for certain vital information. Every such volume is divided into a certain number of 512 byte long sectors. Sector 0 is unused and is available to contain a bootstrap program or other information.

Sector 1 is the *super-block*. The format of a super-block is:

```
/*
 * Structure of the super-block
 */
struct filsys
{
    ushort    s_ysize;           /* size in blocks of i-list */
    daddr_t   s_fsize;          /* size in blocks of entire volume */
    short     s_nfree;          /* number of addresses in s_free */
    daddr_t   s_free[NICFREE];  /* free block list */
    short     s_ninode;         /* number of i-nodes in s_inode */
    ino_t     s_inode[NICINOD]; /* free i-node list */
    char      s_flock;          /* lock during free list manipulation */
    char      s_ilock;          /* lock during i-list manipulation */
    char      s_fmod;           /* super block modified flag */
    char      s_ronly;          /* mounted read-only flag */
    time_t    s_time;           /* last super block update */
    short     s_dinfo[4];       /* device information */
    daddr_t   s_tfree;          /* total free blocks */
    ino_t     s_tinode;         /* total free inodes */
    char      s_fname[6];       /* file system name */
    char      s_fpack[6];       /* file system pack name */
    long      s_fill[13];       /* ADJUST to make sizeof filsys be 512 */
    long      s_magic;          /* magic number to indicate new file system */
    long      s_type;           /* type of new file system */
};
```

```
#define FsMAGIC 0xfd187e20      /* s_magic number */
```

```
#define Fs1b    1              /* 512 byte block */
```

```
#define Fs2b    2              /* 1024 byte block */
```

*S\_type* indicates the file system type. Currently, two types of file systems are supported: the original 512-byte oriented and the new improved 1024-byte oriented. *S\_magic* is used to distinguish the original 512-byte oriented file systems from the newer file systems. If this field is not equal to the magic number, *FsMAGIC*, the type is assumed to be *Fs1b*, otherwise the *s\_type* field is used. In the following description, a block is then determined by the type. For the original 512-byte oriented file system, a block is 512 bytes. For the 1024-byte oriented file system, a block is 1024 bytes or two sectors. The operating system takes care of all conversions from logical block numbers to physical sector numbers.

*S\_ysize* is the address of the first data block after the i-list; the i-list starts just after the super-block, namely in block 2; thus the i-list is *s\_ysize* - 2.

blocks long. *S\_size* is the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block numbers; if an "impossible" block number is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The *s\_free* array contains, in *s\_free*[1], ..., *s\_free*[*s\_nfree*-1], up to 49 numbers of free blocks. *S\_free*[0] is the block number of the head of a chain of blocks constituting the free list. The first long in each free-chain block is the number (up to 50) of free-block numbers listed in the next 50 longs of this chain member. The first of these 50 blocks is the link to the next member of the chain. To allocate a block: decrement *s\_nfree*, and the new block is *s\_free*[*s\_nfree*]. If the new block number is 0, there are no blocks left, so give an error. If *s\_nfree* became 0, read in the block named by the new block number, replace *s\_nfree* by its first word, and copy the block numbers in the next 50 longs into the *s\_free* array. To free a block, check if *s\_nfree* is 50; if so, copy *s\_nfree* and the *s\_free* array into it, write it out, and set *s\_nfree* to 0. In any event set *s\_free*[*s\_nfree*] to the freed block's number and increment *s\_nfree*.

*S\_tfree* is the total free blocks available in the file system.

*S\_ninode* is the number of free i-numbers in the *s\_inode* array. To allocate an i-node: if *s\_ninode* is greater than 0, decrement it and return *s\_inode*[*s\_ninode*]. If it was 0, read the i-list and place the numbers of all free inodes (up to 100) into the *s\_inode* array, then try again. To free an i-node, provided *s\_ninode* is less than 100, place its number into *s\_inode*[*s\_ninode*] and increment *s\_ninode*. If *s\_ninode* is already 100, do not bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

*S\_tinode* is the total free inodes available in the file system.

*S\_flock* and *s\_iloc* are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of *s\_fmmod* on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

*S\_ronly* is a read-only flag to indicate write-protection.

*S\_time* is the last time the super-block of the file system was changed, and is the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). During a reboot, the *s\_time* of the super-block for the root file system is used to set the system's idea of the time.

*S\_fname* is the name of the file system and *s\_fpack* is the name of the pack. I-numbers begin at 1, and the storage for i-nodes begins in block 2. Also, i-nodes are 64 bytes long. I-node 1 is reserved for future use. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. For the format of an inode and its flags, see *inode*(4).

## FILES

/usr/include/sys/filsys.h  
/usr/include/sys/stat.h

## SEE ALSO

fsck(1M), fsdb(1M), mkfs(1M), inode(4).

## NAME

fspec — format specification in text files

## DESCRIPTION

It is sometimes convenient to maintain text files on UNIX with non-standard tabs, (i.e., tabs which are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by UNIX commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets <: and >:. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

- ttabs** The **t** parameter specifies the tab settings for the file. The value of *tabs* must be one of the following:
1. a list of column numbers separated by commas, indicating tabs set at the specified columns;
  2. a **-** followed immediately by an integer *n*, indicating tabs at intervals of *n* columns;
  3. a **-** followed by the name of a "canned" tab specification.

Standard tabs are specified by **t-8**, or equivalently, **t1,9,17,25**, etc. The canned tabs which are recognized are defined by the *tabs(1)* command.

- ssize** The **s** parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.
- mmargin** The **m** parameter specifies a number of spaces to be prepended to each line. The value of *margin* must be an integer.
- d** The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.
- e** The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t-8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

Several UNIX commands correctly interpret the format specification for a file. Among them is *gath* (see *send(1C)*) which may be used to convert files to a standard format acceptable to other UNIX commands.

## SEE ALSO

*ed(1)*, *newform(1)*, *send(1C)*, *tabs(1)*.

## NAME

gettydefs — speed and terminal settings used by getty

## DESCRIPTION

The `/etc/gettydefs` file contains information used by `getty(1M)` (see the *UNIX System Administrator's Manual*) to set up the speed and terminal settings for a line. It supplies information on what the `login` prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing a `<break>` character.

Each entry in `/etc/gettydefs` has the following format:

```
label# initial-flags # final-flags # login-prompt # next-label
```

Each entry is followed by a blank line. The various fields can contain quoted characters of the form `\b`, `\n`, `\c`, etc., as well as `\nnn`, where `nnn` is the octal value of the desired character. The various fields are:

- label* This is the string against which `getty` tries to match its second argument. It is often the speed, such as `1200`, at which the terminal is supposed to run, but it needn't be (see below).
- initial-flags* These flags are the initial `ioctl(2)` settings to which the terminal is to be set if a terminal type is not specified to `getty`. The flags that `getty` understands are the same as the ones listed in `/usr/include/sys/termio.h` (see `termio(7)` in the *UNIX System Administrator's Manual*). Normally only the speed flag is required in the *initial-flags*. `Getty` automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until `getty` executes `login(1)`.
- final-flags* These flags take the same values as the *initial-flags* and are set just prior to `getty` executes `login`. The speed flag is again required. The composite flag `SANE` takes care of most of the other flags that need to be set so that the processor and terminal are communicating in a rational fashion. The other two commonly specified *final-flags* are `TAB3`, so that tabs are sent to the terminal as spaces, and `HUPCL`, so that the line is hung up on the final close.
- login-prompt* This entire field is printed as the *login-prompt*. Unlike the above fields where white space is ignored (a space, tab or new-line), they are included in the *login-prompt* field.
- next-label* If this entry does not specify the desired speed, indicated by the user typing a `<break>` character, then `getty` will search for the entry with *next-label* as its *label* field and set up the terminal for those settings. Usually, a series of speeds are linked together in this fashion, into a closed set. For instance, `2400` linked to `1200`, which in turn is linked to `300`, which finally is linked to `2400`.

If `getty` is called without a second argument, then the first entry of `/etc/gettydefs` is used, thus making the first entry of `/etc/gettydefs` the default entry. It is also used if `getty` can't find the specified *label*. If `/etc/gettydefs` itself is missing, there is one entry built into the command which will bring up a terminal at `300` baud.

It is strongly recommended that after making or modifying `/etc/gettydefs`, it be run through `getty` with the check option to be sure there are no errors.

**FILES**

/etc/gettydefs

**SEE ALSO**

getty(1M), termio(7) in the *UNIX System Administrator's Manual*.  
login(1), ioctl(2).



## NAME

gps — graphical primitive string, format of graphical files

## DESCRIPTION

GPS is a format used to store graphical data. Several routines have been developed to edit and display GPS files on various devices. Also, higher level graphics programs such as *plot* (in *stat(1G)*) and *vtoc* (in *toc(1G)*) produce GPS format output files.

A GPS is composed of five types of graphical data or primitives.

## GPS PRIMITIVES

- lines** The *lines* primitive has a variable number of points from which zero or more connected line segments are produced. The first point given produces a *move* to that location. (A *move* is a relocation of the graphic cursor without drawing.) Successive points produce line segments from the previous point. Parameters are available to set *color*, *weight*, and *style* (see below).
- arc** The *arc* primitive has a variable number of points to which a curve is fit. The first point produces a *move* to that point. If only two points are included a line connecting the points will result, if three points a circular arc through the points is drawn, and if more than three, lines connect the points. (In the future, a spline will be fit to the points if they number greater than three.) Parameters are available to set *color*, *weight*, and *style*.
- text** The *text* primitive draws characters. It requires a single point which locates the center of the first character to be drawn. Parameters are *color*, *font*, *textsize*, and *textangle*.
- hardware** The *hardware* primitive draws hardware characters or gives control commands to a hardware device. A single point locates the beginning location of the *hardware* string.
- comment** A *comment* is an integer string that is included in a GPS file but causes nothing to be displayed. All GPS files begin with a comment of zero length.

## GPS PARAMETERS

- color** *Color* is an integer value set for *arc*, *lines*, and *text* primitives.
- weight** *Weight* is an integer value set for *arc* and *lines* primitives to indicate line thickness. The value 0 is narrow weight, 1 is bold, and 2 is medium weight.
- style** *Style* is an integer value set for *lines* and *arc* primitives to give one of the five different line styles that can be drawn on Tektronix 4010 series storage tubes. They are:
- 0 solid
  - 1 dotted
  - 2 dot dashed
  - 3 dashed
  - 4 long dashed
- font** An integer value set for *text* primitives to designate the text font to be used in drawing a character string. (Currently *font* is expressed as a four-bit *weight* value followed by a four-bit *style* value.)
- textsize** *Textsize* is an integer value used in *text* primitives to express the size of the characters to be drawn. *Textsize* represents the height of characters in absolute *universe-units* and is stored at one-fifth

this value in the size-orientation (*so*) word (see below).

**textangle** *Textangle* is a signed integer value used in *text* primitives to express rotation of the character string around the beginning point. *Textangle* is expressed in degrees from the positive x-axis and can be a positive or negative value. It is stored in the size-orientation (*so*) word as a value 256/360 of it's absolute value.

#### ORGANIZATION

GPS primitives are organized internally as follows:

<b>lines</b>	<i>cw points sw</i>
<b>arc</b>	<i>cw points sw</i>
<b>text</b>	<i>cw point sw so [string]</i>
<b>hardware</b>	<i>cw point [string]</i>
<b>comment</b>	<i>cw [string]</i>

**cw** *Cw* is the control word and begins all primitives. It consists of four bits that contain a primitive-type code and twelve bits that contain the word-count for that primitive.

**point(s)** *Point(s)* is one or more pairs of integer coordinates. *Text* and *hardware* primitives only require a single *point*. *Point(s)* are values within a Cartesian plane or *universe* having 64K (-32K to +32K) points on each axis.

**sw** *Sw* is the style-word and is used in *lines*, *arc*, and *text* primitives. The first eight bits contain *color* information. In *arc* and *lines* the last eight bits are divided as four bits *weight* and four bits *style*. In the *text* primitive the last eight bits of *sw* contain the *font*.

**so** *So* is the size-orientation word used in *text* primitives. The first eight bits contain text size and the remaining eight bits contain text rotation.

**string** *String* is a null-terminated character string. If the string does not end on a word boundary an additional null is added to the GPS file to insure word-boundary alignment.

#### SEE ALSO

graphics(1G).

**NAME**

group — group file

**DESCRIPTION**

*Group* contains for each group the following information:

- group name
- encrypted password
- numerical group ID
- comma-separated list of all user allowed in the group

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory */etc*. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

**FILES**

*/etc/group*

**SEE ALSO**

*newgrp(1)*, *passwd(1)*, *crypt(3C)*, *passwd(4)*.

## NAME

inittab — script for the init process

## DESCRIPTION

The *inittab* file supplies the script to *init*'s role as a general process dispatcher. The process that constitutes the majority of *init*'s process dispatching activities is the line process */etc/getty* that initiates individual terminal lines. Other processes typically dispatched by *init* are daemons and the shell.

The *inittab* file is composed of entries that are position dependent and have the following format:

```
id:rstate:action:process
```

Each entry is delimited by a newline, however, a backslash (\) preceding a newline indicates a continuation of the entry. Up to 512 characters per entry are permitted. Comments may be inserted in the *process* field using the *sh*(1) convention for comments. Comments for lines that spawn *gettys* are displayed by the *who*(1) command. It is expected that they will contain some information about the line such as the location. There are no limits (other than maximum entry size) imposed on the number of entries within the *inittab* file. The entry fields are:

- id** This is one or two characters used to uniquely identify an entry.
- rstate** This defines the *run-level* in which this entry is to be processed. *Run-levels* effectively correspond to a configuration of processes in the system. That is, each process spawned by *init* is assigned a *run-level* or *run-levels* in which it is allowed to exist. The *run-levels* are represented by a number ranging from 0 through 6. As an example, if the system is in *run-level* 1, only those entries having a 1 in the *rstate* field will be processed. When *init* is requested to change *run-levels*, all processes which do not have an entry in the *rstate* field for the target *run-level* will be sent the warning signal (SIGTERM) and allowed a 20 second grace period before being forcibly terminated by a kill signal (SIGKILL). The *rstate* field can define multiple *run-levels* for a process by selecting more than one *run-level* in any combination from 0-6. If no *run-level* is specified, then the process is assumed to be valid at all *run-levels* 0-6. There are three other values, a, b and c, which can appear in the *rstate* field, even though they are not true *run-levels*. Entries which have these characters in the *rstate* field are processed only when the *telinit* (see *init*(1M)) process requests them to be run (regardless of the current *run-level* of the system). They differ from *run-levels* in that *init* can never enter *run-level* a, b or c. Also, a request for the execution of any of these processes does not change the current *run-level*. Furthermore, a process started by an a, b or c command is not killed when *init* changes levels. They are only killed if their line in */etc/inittab* is marked off in the *action* field, their line is deleted entirely from */etc/inittab*, or *init* goes into the *SINGLE USER* state.
- action** Key words in this field tell *init* how to treat the process specified in the *process* field. The actions recognized by *init* are as follows:
- respawn** If the process does not exist then start the process, do not wait for its termination (continue scanning the *inittab* file), and when it dies restart the process. If the process currently exists then do nothing and continue scanning the *inittab* file.

- wait** Upon *init*'s entering the *run-level* that matches the entry's *rstate*, start the process and wait for its termination. All subsequent reads of the *inittab* file while *init* is in the same *run-level* will cause *init* to ignore this entry.
- once** Upon *init*'s entering a *run-level* that matches the entry's *rstate*, start the process, do not wait for its termination and when it dies, do not restart the process. If upon entering a new *run-level*, where the process is still running from a previous *run-level* change, the program will not be restarted.
- boot** The entry is to be processed only at *init*'s boot-time read of the *inittab* file. *Init* is to start the process, not wait for its termination, and when it dies, not restart the process. In order for this instruction to be meaningful, the *rstate* should be the default or it must match *init*'s *run-level* at boot time. This action is useful for an initialization function following a hardware reboot of the system.
- bootwait** The entry is to be processed only at *init*'s boot-time read of the *inittab* file. *Init* is to start the process, wait for its termination and, when it dies, not restart the process.
- powerfail** Execute the process associated with this entry only when *init* receives a power fail signal (SIGPWR see *signal(2)*).
- powerwait** Execute the process associated with this entry only when *init* receives a power fail signal (SIGPWR) and wait until it terminates before continuing any processing of *inittab*.
- off** If the process associated with this entry is currently running, send the warning signal (SIGTERM) and wait 20 seconds before forcibly terminating the process via the kill signal (SIGKILL). If the process is nonexistent, ignore the entry.
- ondemand** This instruction is really a synonym for the **respawn** action. It is functionally identical to **respawn** but is given a different keyword in order to divorce its association with *run-levels*. This is used only with the **a**, **b** or **c** values described in the *rstate* field.
- initdefault** An entry with this *action* is only scanned when *init* initially invoked. *Init* uses this entry, if it exists, to determine which *run-level* to enter initially. It does this by taking the highest *run-level* specified in the *rstate* field and using that as its initial state. If the *rstate* field is empty, this is interpreted as **0123456** and so *init* will enter *run-level* 6. Also, the **initdefault** entry cannot specify that *init* start in the *SINGLE USER* state. Additionally, if *init* doesn't find an **initdefault** entry in */etc/inittab*, then it will request an initial *run-level* from the user at reboot time.
- sysinit** Entries of this type are executed before *init* tries to access the console. It is expected that this entry will

be only used to initialize devices on which *init* might try to ask the *run-level* question. These entries are executed and waited for before continuing.

*process* This is a *sh* command to be executed. The entire **process** field is prefixed with *exec* and passed to a forked *sh* as *sh -c 'exec command'*. For this reason, any legal *sh* syntax can appear in the *process* field. Comments can be inserted with the ; *#comment* syntax.

**FILES**

/etc/inittab

**SEE ALSO**

getty(1M), init(1M) in the *UNIX System Administrator's Manual*.  
sh(1), who(1), exec(2), open(2), signal(2).

**NAME**

inode — format of an inode

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/ino.h>
```

**DESCRIPTION**

An i-node for a plain file or directory in a file system has the following structure defined by `<sys/ino.h>`.

```
/* Inode structure as it appears on a disk block. */
struct dinode
{
    ushort di_mode; /* mode and type of file */
    short di_nlink; /* number of links to file */
    ushort di_uid; /* owner's user id */
    ushort di_gid; /* owner's group id */
    off_t di_size; /* number of bytes in file */
    char di_addr[40]; /* disk block addresses */
    time_t di_atime; /* time last accessed */
    time_t di_mtime; /* time last modified */
    time_t di_ctime; /* time created */
};
/*
 * the 40 address bytes:
 * 39 used; 13 addresses
 * of 3 bytes each.
 */
```

For the meaning of the defined types `off_t` and `time_t` see `types(5)`.

**FILES**

`/usr/include/sys/ino.h`

**SEE ALSO**

`stat(2)`, `fs(4)`, `types(5)`.

**NAME**

issue — issue identification file

**DESCRIPTION**

The file `/etc/issue` contains the *issue* or project identification to be printed as a login prompt. This is an ASCII file which is read by program *getty* and then written to any terminal spawned or respawned from the *lines* file.

**FILES**

`/etc/issue`

**SEE ALSO**

`login(1)`.



**NAME**

ldfcn — common object file access routines

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

**DESCRIPTION**

The common object file access routines are a collection of functions for reading an object file that is in VAX or 3B20S (common) object file form. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

The interface between the calling program and the object file access routines is based on the defined type **LDFILE**, defined as **struct ldfile**, declared in the header file **ldfcn.h**. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The function **ldopen(3X)** allocates and initializes the **LDFILE** structure and returns a pointer to the structure to the calling program. The fields of the **LDFILE** structure may be accessed individually through macros defined in **ldfcn.h** and contain the following information:

**LDFILE \*ldptr;**

**TYPE(ldptr)** The file magic number, used to distinguish between archive members and simple object files.

**OPTR(ldptr)** The file pointer returned by *lopen* and used by the standard input/output functions.

**OFFSET(ldptr)** The file address of the beginning of the object file; the offset is non-zero if the object file is a member of an archive file.

**HEADER(ldptr)** The file header structure of the object file.

The object file access functions themselves may be divided into four categories:

- (1) functions that open or close an object file

```
ldopen(3X) and ldaopen
  open a common object file
ldclose(3X) and ldaclose
  close a common object file
```

- (2) functions that read header or symbol table information

```
ldahread(3X)
  read the archive header of a member of an archive
  file
ldfhread(3X)
  read the file header of a common object file
ldshread(3X) and ldnshread
  read a section header of a common object file
ldtbread(3X)
  read a symbol table entry of a common object file
```

- (3) functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular

section.

*ldohseek*(3X)

seek to the optional file header of a common object file

*ldsseek*(3X) and *ldnsseek*

seek to a section of a common object file

*ldrseek*(3X) and *ldnrseek*

seek to the relocation information for a section of a common object file

*ldlseek*(3X) and *ldnlseek*

seek to the line number information for a section of a common object file

*ldtbseek*(3X)

seek to the symbol table of a common object file

(4) the function *ldtbindx*(3X) which returns the index of a particular common object file symbol table entry

These functions are described in detail in their respective manual pages.

All the functions except *ldopen*, *ldaopen* and *ldtbindx* return either SUCCESS or FAILURE, both constants defined in *ldfcn.h*. *Ldopen* and *ldaopen* both return pointers to a LDFILE structure.

## MACROS

Additional access to an object file is provided through a set of macros defined in *ldfcn.h*. These macros parallel the standard input/output file reading and manipulating functions, translating a reference of the LDFILE structure into a reference to its file descriptor field.

The following macros are provided:

LDFILE       \*ldptr;

GETC(ldptr)

FGETC(ldptr)

GETW(ldptr)

UNGETC(c, ldptr)

FGETS(s, n, ldptr)

FREAD((char \*) ptr, sizeof (\*ptr), nitems, ldptr)

FSEEK(ldptr, offset, ptrname)

FTELL(ldptr)

REWIND(ldptr)

FEOF(ldptr)

FERROR(ldptr)

FILENO(ldptr)

SETBUF(ldptr, buf)

See the manual entries for the corresponding standard input/output library functions for details on the use of these macros.

The program must be loaded with the object file access routine library *libld.a*.

## CAVEAT

The macro FSEEK defined in the header file *ldfcn.h* translates into a call to the standard input/output function *fseek*(3S). FSEEK should not be used to seek from the end of an archive file since the end of an archive file may not be the same as the end of one of its object file members!

## SEE ALSO

*fseek*(3S), *ldahread*(3X), *ldclose*(3X), *ldfhread*(3X), *ldlread*(3X),

ldlseek(3X), ldohseek(3X), ldopen(3X), ldrseek(3X), ldlseek(3X),  
ldhread(3X), ldtbindex(3X), ldtbread(3X), ldtbseek(3X).  
*Common Object File Format*, by I. S. Law.

**NAME**

linenum — line number entries in a common object file

**SYNOPSIS**

```
#include <linenum.h>
```

**DESCRIPTION**

Compilers based on *pcc* generate an entry in the object file for each C source line on which a breakpoint is possible (when invoked with the *-g* option; see *cc(1)*). Users can then reference line numbers when using the appropriate software test system (see *sdb(1)*). The structure of these line number entries appears below.

```
struct lineno
{
    union
    {
        long    l_symndx ;
        long    l_paddr ;
    }          l_addr ;
    unsigned short l_inno ;
};
```

Numbering starts with one for each function. The initial line number entry for a function has *l\_inno* equal to zero, and the symbol table index of the function's entry is in *l\_symndx*. Otherwise, *l\_inno* is non-zero, and *l\_paddr* is the physical address of the code for the referenced line. Thus the overall structure is the following:

<i>l_addr</i>	<i>l_inno</i>
function symtab index	0
physical address	line
physical address	line
...	
function symtab index	0
physical address	line
physical address	line
...	

**SEE ALSO**

*cc(1)*, *sdb(1)*, *a.out(4)*.

## NAME

master — master device information table

## DESCRIPTION

This file is used by the *config(1M)* program to obtain device information that enables it to generate the configuration files. The file consists of 3 parts, each separated by a line with a dollar sign (\$) in column 1. Part 1 contains device information; part 2 contains names of devices that have aliases; part 3 contains tunable parameter information. Any line with an asterisk (\*) in column 1 is treated as a comment.

Part 1 contains lines consisting of at least 10 fields and at most 13 fields, with the fields delimited by tabs and/or blanks:

- Field 1: device name (8 chars. maximum).
- Field 2: interrupt vector size (decimal, in bytes).
- Field 3: device mask (octal)—each "on" bit indicates that the handler exists:
  - 000100 initialization handler
  - 000040 power-failure handler
  - 000020 open handler
  - 000010 close handler
  - 000004 read handler
  - 000002 write handler
  - 000001 ioctl handler.
- Field 4: device type indicator (octal):
  - 000400 VAX-11/780 massbus adapter
  - 000200 allow only one of these devices
  - 000100 suppress count field in the *conf.c* file
  - 000040 suppress interrupt vector
  - 000020 required device
  - 000010 block device
  - 000004 character device
  - 000002 floating vector
  - 000001 fixed vector.
- Field 5: handler prefix (4 chars. maximum).
- Field 6: device address size (decimal).
- Field 7: major device number for block-type device.
- Field 8: major device number for character-type device.
- Field 9: maximum number of devices per controller (decimal).
- Field 10: maximum bus request level (4 through 7).
- Fields 11-13: optional configuration table structure declarations (8 chars. maximum).

Part 2 contains lines with 2 fields each:

- Field 1: alias name of device (8 chars. maximum).
- Field 2: reference name of device (8 chars. maximum; specified in part 1).

Part 3 contains lines with 2 or 3 fields each:

- Field 1: parameter name (as it appears in description file; 20 chars. maximum)
- Field 2: parameter name (as it appears in the *conf.c* file; 20 chars. maximum)
- Field 3: default parameter value (20 chars. maximum; parameter specification is required if this field is omitted)

Devices that are not interrupt-driven have an interrupt vector size of zero. The 040 bit in Field 4 causes *config(1M)* to record the interrupt vector although the *low.s* (*univec.c* on the VAX-11/780) file will show no interrupt vector assignment at those locations (interrupts here will be treated as strays).

**SEE ALSO**

*config(1M)*.

**NAME**

master — master device information table

**DESCRIPTION**

This file is used by the *config(1M)* program to obtain device information that enables it to generate the configuration file. *Master* contains lines of various forms for controlling the configuration of hardware devices, software drivers, parameters and aliases.

Hardware devices and software drivers are defined as follows:

- Field 1: device name (8 chars maximum).
- Field 2: element type (*dev*, *mhd*, *pc* or *sw*)
- Field 3: functions for this device:
  - o* open handler
  - c* close handler
  - r* read handler
  - w* write handler
  - i* ioctl handler
  - d* diagnostic handler
  - s* startup routine
  - f* fork
  - e* exec
  - x* exit
- Field 4: element characteristics:
  - o* specify only once
  - s* suppress count field
  - r* required device
  - b* block device
  - c* character device
- Field 5: handler prefix
- Field 6: major device number if block-type device
- Field 7: major device number if character-type device
- Field 8: number of sub-devices per device
- Field 9: diagnostic port number if diagnosable device
- Field 10: configuration table structure

Parameters are defined as follows:

- Field 1: parameter name
- Field 2: element type (*param*)
- Field 3: element characteristics, as defined above
- Field 4: parameter name to appear in *conf.c* file

UNIX devices and UNIX devices with arguments are defined as follows:

- Field 1: device name
- Field 2: element type (*udev* or *udeva*)
- Field 3: element characteristics, as defined above
- Field 4: device name to appear in *conf.c* file

Aliases for names are defined as follows:

- Field 1: alias name
- Field 2: element type (*alias*)
- Field 3: reference name of device

Lines to be ignored by the *config* program, but are necessary to the diagnostic system, are defined as follows:

Field 1: name to be ignored

Field 2: element type (ignore)

**SEE ALSO**

*config*(1M) *sysdef*(1M).



**NAME**

mnttab — mounted file system table

**SYNOPSIS**

```
#include <mnttab.h>
```

**DESCRIPTION**

*Mnttab* resides in directory */etc* and contains a table of devices, mounted by the *mount(1M)* command, in the following structure as defined by *<mnttab.h>*:

```
struct mnttab {
    char      mt_dev[10];
    char      mt_filsys[10];
    short     mt_ro_flg;
    time_t    mt_time;
};
```

Each entry is 26 bytes in length; the first 10 bytes are the null-padded name of the place where the *special file* is mounted; the next 10 bytes represent the null-padded root name of the mounted special file; the remaining 6 bytes contain the mounted *special file*'s read/write permissions and the date on which it was mounted.

The maximum number of entries in *mnttab* is based on the system parameter *NMOUNT* located in */usr/src/uts/ct/conf.c*, which defines the number of allowable mounted special files.

**SEE ALSO**

*mount(1M)*, *setmnt(1M)*.

**NAME**

passwd — password file

**DESCRIPTION***Passwd* contains for each user the following information:

login name  
 encrypted password  
 numerical user ID  
 numerical group ID  
 GCOS job number, box number, optional GCOS user ID  
 initial working directory  
 program to use as Shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The GCOS field is used only when communicating with that system, and in other installations can contain any desired information. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, the Shell itself is used.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user ID's to names.

The encrypted password consists of 13 characters chosen from a 64 character alphabet (., /, 0-9, A-Z, a-z), except when the password is null in which case the encrypted password is also null. Password aging is effected for a particular user if his encrypted password in the password file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the super-user.)

The first character of the age, *M* say, denotes the maximum number of weeks for which a password is valid. A user who attempts to login after his password has expired will be forced to supply a new one. The next character, *m* say, denotes the minimum period in weeks which must expire before the password may be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.) *M* and *m* have numerical values in the range 0-63 that correspond to the 64 character alphabet shown above (i.e. / = 1 week; z = 63 weeks). If  $m = M = 0$  (derived from the string . or ..) the user will be forced to change his password the next time he logs in (and the "age" will disappear from his entry in the password file). If  $m > M$  (signified, e.g., by the string ./) only the super-user will be able to change the password.

**FILES**

/etc/passwd

**SEE ALSO**

login(1), passwd(1), a64l(3C), crypt(3C), getpwent(3C), group(4).

## NAME

plot — graphics interface

## DESCRIPTION

Files of this format are produced by routines described in *plot(3X)* and are interpreted for various devices by commands described in *tplot(1G)*. A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the x and y values; each value is a signed integer. The last designated point in an *l*, *m*, *n*, or *p* instruction becomes the "current point" for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in *plot(3X)*.

- m** move: The next four bytes give a new current point.
- n** cont: Draw a line from the current point to the point given by the next four bytes. See *tplot(1G)*.
- p** point: Plot the point given by the next four bytes.
- l** line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.
- t** label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a new-line.
- e** erase: Start another frame of output.
- f** linemod: Take the following string, up to a new-line, as the style for drawing further lines. The styles are "dotted", "solid", "longdashed", "shortdashed", and "dotdashed". Effective only for the *-T4014* and *-Tver* options of *tplot(1G)* (Tektronix 4014 terminal and Versatec plotter).
- s** space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *tplot(1G)*. The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

DASI 300	space(0, 0, 4096, 4096);
DASI 300s	space(0, 0, 4096, 4096);
DASI 450	space(0, 0, 4096, 4096);
Tektronix 4014	space(0, 0, 3120, 3120);
Versatec plotter	space(0, 0, 2048, 2048);

## SEE ALSO

graph(1G), tplot(1G), plot(3X), gps(4), term(5).

**NAME**

pnch — file format for card images

**DESCRIPTION**

The PNCH format is a convenient representation for files consisting of card images in an arbitrary code.

A PNCH file is a simple concatenation of card records. A card record consists of a single control byte followed by a variable number of data bytes. The control byte specifies the number (which must lie in the range 0-80) of data bytes that follow. The data bytes are 8-bit codes that constitute the card image. If there are fewer than 80 data bytes, it is understood that the remainder of the card image consists of trailing blanks.

**SEE ALSO**

send(1C).

**NAME**

profile — setting up an environment at login time

**DESCRIPTION**

If your login directory contains a file named **.profile**, that file will be executed (via the shell's **exec .profile**) before your session begins; **.profiles** are handy for setting exported environment variables and terminal modes. If the file **/etc/profile** exists, it will be executed for every user before the **.profile**. The following example is typical (except for the comments):

```
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 22
# Tell me when new mail comes in
MAIL=/usr/mail/myname
# Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Set terminal type
echo "terminal: \c"
read TERM
case $TERM in
    300)          stty cr2 n0 tabs; tabs;;
    300s)         stty cr2 n0 tabs; tabs;;
    450)          stty cr2 n0 tabs; tabs;;
    hp)           stty cr0 n0 tabs; tabs;;
    745|735)      stty cr1 n1 -tabs; TERM=745;;
    43)           stty cr1 n0 -tabs;;
    4014|tek)     stty cr0 n0 -tabs ff1; TERM=4014; echo "\33";;
    *)           echo "$TERM unknown";;
esac
```

**FILES**

\$HOME/.profile  
/etc/profile

**SEE ALSO**

env(1), login(1), mail(1), sh(1), stty(1), su(1), environ(5), term(5).

**NAME**

reloc — relocation information for a common object file

**SYNOPSIS**

```
#include <reloc.h>
```

**DESCRIPTION**

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format.

```
struct reloc
{
    long    r_vaddr ; /* (virtual) address of reference */
    long    r_symndx ; /* index into symbol table */
    short   r_type ; /* relocation type */
};

/*
 * All generics
 *   reloc. already performed to symbol in the same section
 */
#define R_ABS          0

/*
 * 3B generic
 *   24-bit direct reference
 *   24-bit "relative" reference
 *   16-bit optimized "indirect" TV reference
 *   24-bit "indirect" TV reference
 *   32-bit "indirect" TV reference
 */
#define R_DIR24    04
#define R_REL24    05
#define R_OPT16    014
#define R_IND24    015
#define R_IND32    016

/*
 * DEC Processors VAX 11/780 and VAX 11/750
 */
#define R_RELBYTE    017
#define R_RELWORD    020
#define R_RELLONG    021
#define R_PCRBYTE    022
#define R_PCRWORD    023
#define R_PCRLONG    024
```

As the link editor reads each input section and performs relocation, the relocation entries are read. They direct how references found within the input section are treated.

**R\_ABS**      The reference is absolute, and no relocation is necessary. The entry will be ignored.

- R\_DIR24** A direct, 24-bit reference to a symbol's virtual address.
- R\_REL24** A "PC-relative", 24-bit reference to a symbol's virtual address. Relative references occur in instructions such as jumps and calls. The actual address used is obtained by adding a constant to the value of the program counter at the time the instruction is executed.
- R\_OPT16** An optimized, indirect, 16-bit reference through a transfer vector. The instruction contains the offset into the transfer vector table to the transfer vector where the actual address of the referenced word is stored.
- R\_IND24** An indirect, 24-bit reference through a transfer vector. The instruction contains the virtual address of the transfer vector, where the actual address of the referenced word is stored.
- R\_IND32** An indirect, 32-bit reference through a transfer vector. The instruction contains the virtual address of the transfer vector, where the actual address of the referenced word is stored.
- R\_RELBYTE**  
A direct 8 bit reference to a symbol's virtual address.
- R\_RELWORD**  
A direct 16 bit reference to a symbol's virtual address.
- R\_RELLONG**  
A direct 32 bit reference to a symbol's virtual address.
- R\_PCRBYTE**  
A "PC-relative", 8 bit reference to a symbol's virtual address.
- R\_PCRWORD**  
A "PC-relative", 16 bit reference to a symbol's virtual address.
- R\_PCRLONG**  
A "PC-relative", 32 bit reference to a symbol's virtual address.

On the VAX processors relocation of a symbol index of -1 indicates that the relative difference between the current segment's start address and the program's load address is added to the relocatable address.

Other relocation types will be defined as they are needed.

Relocation entries are generated automatically by the assembler and automatically utilized by the link editor. A link editor option exists for removing the relocation entries from an object file.

**SEE ALSO**

ld(1), strip(1), a.out(4), syms(4).

## NAME

sccsfile — format of SCCS file

## DESCRIPTION

An SCCS file is an ASCII file. It consists of six logical parts: the *checksum*, the *delta table* (contains information about each delta), *user names* (contains login names and/or numerical group IDs of users who may add deltas), *flags* (contains definitions of internal keywords), *comments* (contains arbitrary descriptive information about the file), and the *body* (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as *the control character* and will be represented graphically as @. Any line described below which is not depicted as beginning with the control character is prevented from beginning with the control character.

Entries of the form DDDDD represent a five digit string (a number between 00000 and 99999).

Each logical part of an SCCS file is described in detail below.

*Checksum*

The checksum is the first line of an SCCS file. The form of the line is:

**@hDDDDDD**

The value of the checksum is the sum of all characters, except those of the first line. The @h provides a *magic number* of (octal) 064001.

*Delta table*

The delta table consists of a variable number of entries of the form:

```

@s DDDDD/DDDD/DDDD
@d <type> <SCCS ID> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
.
.
@c <comments> ...
.
.
@e

```

The first line (@s) contains the number of lines inserted/deleted/unchanged respectively. The second line (@d) contains the type of the delta (currently, normal: D, and removed: R), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The @i, @x, and @g lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.



The **@m** lines (optional) each contain one MR number associated with the delta; the **@c** lines contain comments associated with the delta.

The **@e** line ends the delta table entry.

#### User names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines **@u** and **@U**. An empty list allows anyone to make a delta.

#### Flags

Keywords used internally (see *admin(1)* for more information on their use). Each flag line takes the form:

```
@f <flag>    <optional text>
```

The following flags are defined:

```
@f t  <type of program>
@f v  <program name>
@f i
@f b
@f m  <module name>
@f f  <floor>
@f c  <ceiling>
@f d  <default-sid>
@f n
@f j
@f l  <lock-releases>
@f q  <user defined>
@f z  <reserved for use in interfaces>
```

The **t** flag defines the replacement for the **%Y%** identification keyword. The **v** flag controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program. The **i** flag controls the warning/error aspect of the "No id keywords" message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message will cause a "fatal" error (the file will not be gotten, or the delta will not be made). When the **b** flag is present the **-b** keyletter may be used on the *get* command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the **%M%** identification keyword. The **f** flag defines the "floor" release; the release below which no deltas may be added. The **c** flag defines the "ceiling" release; the release above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a *get* command. The **n** flag causes *delta* to insert a "null" delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (e.g., when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes *get* to allow concurrent edits of the same base SID. The **l** flag defines a *list* of releases that are *locked* against editing (*get(1)* with the **-e** keyletter). The **q** flag defines the replacement for the **%Q%** identification keyword. **z** flag

is used in certain specialized interface programs.

**Comments**

Arbitrary text surrounded by the bracketing lines @t and @T. The comments section typically will contain a description of the file's purpose.

**Body**

The body consists of text lines and control lines. Text lines don't begin with the control character, control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, represented by:

```
@I DDDDD
@D DDDDD
@E DDDDD
```

respectively. The digit string is the serial number corresponding to the delta for the control line.

**SEE ALSO**

admin(1), delta(1), get(1), prs(1).

*Source Code Control System User's Guide* in the *UNIX System User's Guide*.

**NAME**

scnhdr — section header for a common object file

**SYNOPSIS**

```
#include <scnhdr.h>
```

**DESCRIPTION**

Every common object file has a table of section headers to specify the layout of the data within the file. Each section within an object file has its own header. The C structure appears below.

```
struct scnhdr
{
    char          s_name[SYMNMLEN]; /* section name */
    long         s_paddr; /* physical address */
    long         s_vaddr; /* virtual address */
    long         s_size; /* section size */
    long         s_scnptr; /* file ptr to raw data */
    long         s_relptr; /* file ptr to relocation */
    long         s_innoptr; /* file ptr to line numbers */
    unsigned short s_nreloc; /* # reloc entries */
    unsigned short s_nlnno; /* # line number entries */
    long         s_flags; /* flags */
};
```

File pointers are byte offsets into the file; they can be used as the offset in a call to *fseek*(3S). If a section is initialized, the file contains the actual bytes. An uninitialized section is somewhat different. It has a size, symbols defined in it, and symbols that refer to it. But it can have no relocation entries, line numbers, or data. Consequently, an uninitialized section has no raw data in the object file, and the values for *s\_scnptr*, *s\_relptr*, *s\_innoptr*, *s\_nreloc*, and *s\_nlnno* are zero.

**SEE ALSO**

ld(1), *fseek*(3S), a.out(4).

**NAME**

syms — common object file symbol table format

**SYNOPSIS****#include** <syms.h>**DESCRIPTION**

Common object files contain information to support *symbolic* software testing (see *sdb*(1). Line number entries, *linenum*(4), and extensive symbolic information permit testing at the C *source* level. Every object file's symbol table is organized as shown below.

File name 1.

Function 1.

Local symbols for function 1.

Function 2

Local symbols for function 2.

...

Static externs for file 1.

File name 2.

Function 1.

Local symbols for function 1.

Function 2.

Local symbols for function 2.

...

Static externs for file 2.

...

Defined global symbols.

Undefined global symbols.

The entry for a symbol is a fixed-length structure. The members of the structure hold the name (null padded), its value, and other information. The C structure is given below.

```
#define SYMNMLEN 8
#define FILNMLEN 14
```

```
struct  syment
{
    char                n_name[SYMNMLEN] ;
    long               n_value ;/* value of symbol */
    short             n_scnum ;/* section number */
    unsigned short n_type ;    /* type and derived type */
    char              n_sclass ;/* storage class */
    char              n_numaux ;/* number of aux entries */
};
```

Meaningful values and explanations for them are given in both *syms.h* and *Common Object File Format*. Anyone who needs to interpret the entries should seek more information in these sources. Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```

union auxent
{
    struct
    {
        long          x_tagndx;
        union
        {
            struct
            {
                unsigned short x_inno;
                unsigned short x_size;
            } x_insz;
            long          x_fsize;
        } x_misc;
        union
        {
            struct
            {
                long          x_innoptr;
                long          x_endndx;
            } x_fcn;
            struct
            {
                unsigned short x_dimen[DIMNUM];
            } x_ary;
        } x_fcrary;
        unsigned short x_tvndx;
    } x_sym;
    struct
    {
        char          x_fname[FILNMLEN];
    } x_file;
    struct
    {
        long          x_scnlen;
        unsigned short x_nreloc;
        unsigned short x_nlinno;
    } x_scn;

    struct
    {
        unsigned short x_tvlen;
        unsigned short x_tvran[2];
    } x_tv;
};

```

Indexes of symbol table entries begin at zero.

**SEE ALSO**

sdb(1), a.out(4), linenum(4).

*Common Object File Format* by I. S. Law.

**NAME**

system -- format of 3B20S system description file

**DESCRIPTION**

This file contains information about the hardware configuration and system-dependent parameters for the user's system. A more complete description of the system file is found in Setting up UNIX in the *UNIX System Administrator's Guide*. This information is used by the *config(1M)* program in configuring systems. The file is divided into two sections, separated by a line with a dollar sign (\$) in column 1. The first section describes the hardware configuration and the second contains system-dependent information. Any lines with a number sign (#) in column 1 are treated as comments and are ignored. Blank lines are also ignored. All fields may be separated by one or more space and tab characters.

The following codes are used throughout the following description:

<i>Name</i>	<i>Meaning</i>
chan	channel
count	number of disk blocks in swap or dump area
dev	device on a channel
devname	name of device
driver	name of a software device driver
equip	equipage
hv	hardware version
inter	interrupt source bit
low	lowest disk block in swap or dump area
minor	minor device number
mt	maintenance type
mv	maintenance version
num	the number of instances of a software device driver
parm	name of a UNIX parameter
pc	name of device driver for a PC
pumpcode	path name of pump code file
slot	slot number of a sub-device on its device
unit	logical unit number of a device
value	value of a UNIX parameter

**Hardware Configuration**

This section describes the configuration of the Control Unit (CU) and its components, the Disk File Controllers (DFCs) and their Moving Head Disks (MHDs), and the Input Output Processors (IOPs) and their Peripheral Controllers (PCs). Any line that describes an IOP, DFC, MHD or PC may optionally have an exclamation point (!) preceding the first field. This indicates that a device should not automatically be brought into service by the system (see *don(1M)*). Note that an exclamation point which precedes an IOP implies that neither the IOP nor its PCs will be brought into service. The same applies to a DFC and its MHDs.

The CU and its components are specified as follows:

cu	unit	chan	dev	mt	mv	hv		
	cc	unit	mt	mv	hv	equip	0	
	masc	unit	mt	mv	hv	equip	0	
	sat	unit	mt	mv	hv	equip	0	
	ch	unit	mt	mv	hv	equip	0	
	ch	unit	mt	mv	hv	equip	0	
	csu	unit	mt	mv	hv	equip	0	
	dma	unit	mt	mv	hv	equip	0	
		ch	unit	mt	mv	hv	equip	inter

Each DFC and its MHDs are specified as follows:

```

dfc  unit  chan  dev  mt  hv  mv  equip
     mhd  unit  slot  mt  hv  mv  equip

```

Each IOP and its PCs are specified as follows:

```

iop  unit  chan  dev  mt  mv  hv  equip  [pumpcode]
     pc  unit  slot  mt  mv  hv  equip

```

### System-Dependent Information

This section specifies UNIX devices, UNIX parameters and software drivers.

The root and pipe devices are specified by:

```

root  devname  minor
pipe  devname  minor

```

The swap and dump devices are specified by:

```

swap  devname  minor  low  count
dump  devname  minor  low  count

```

Tunable parameters are specified by:

```

parm  value

```

Software drivers are specified in one of two forms:

```

driver  num
driver

```

### SEE ALSO

config(1M), don(1M), master(4).

Setting up UNIX in the *UNIX System Administrator's Guide*.

**NAME**

utmp, wtmp — utmp and wtmp entry formats

**SYNOPSIS**

```
#include <sys/types.h>
#include <utmp.h>
```

**DESCRIPTION**

These files, which hold user and accounting information for such commands as *who*(1), *write*(1), and *login*(1), have the following structure as defined by <utmp.h>:

```
#define  UTMP_FILE    "/etc/utmp"
#define  WTMP_FILE    "/etc/wtmp"
#define  ut_name      ut_user

struct utmp {
    char    ut_user[8];          /* User login name */
    char    ut_id[4];           /* /etc/inittab id (usually line #) */
    char    ut_line[12];        /* device name (console, lnxx) */
    short   ut_pid;             /* process id */
    short   ut_type;            /* type of entry */
    struct  exit_status {
        short   e_termination; /* Process termination status */
        short   e_exit;         /* Process exit status */
    } ut_exit;                  /* The exit status of a process
                               * marked as DEAD_PROCESS. */
    time_t  ut_time;           /* time entry was made */
};

/* Definitions for ut_type */
#define EMPTY          0
#define RUN_LVL        1
#define BOOT_TIME      2
#define OLD_TIME       3
#define NEW_TIME       4
#define INIT_PROCESS   5          /* Process spawned by "init" */
#define LOGIN_PROCESS  6          /* A "getty" process waiting for login */
#define USER_PROCESS   7          /* A user process */
#define DEAD_PROCESS   8
#define ACCOUNTING     9
#define UTMAXTYPE      ACCOUNTING /* Largest legal value of ut_type */

/* Special strings or formats used in the "ut_line" field when */
/* accounting for something other than a process. */
/* No string for the ut_line field can be more than 11 chars + */
/* a NULL in length. */
#define RUNLVL_MSG     "run-level %c"
#define BOOT_MSG       "system boot"
#define OTIME_MSG      "old time"
#define NTIME_MSG      "new time"
```

**FILES**

```
/usr/include/utmp.h
/etc/utmp
/etc/wtmp
```

**SEE ALSO**

*login*(1), *who*(1), *write*(1), *getut*(3C).





**NAME**

intro — introduction to miscellany

**DESCRIPTION**

This section describes miscellaneous facilities such as macro packages, character set tables, etc.

**NAME**

ascii — map of ASCII character set

**SYNOPSIS**

cat /usr/pub/ascii

**DESCRIPTION**

*Ascii* is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. It contains:

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dcl	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 \$	045 %	046 &	047 ^
050 (	051 )	052 *	053 +	054 ,	055 -	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135 ]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174	175 }	176 ~	177 del

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dcl	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 \$	25 %	26 &	27 ^
28 (	29 )	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d ]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c	7d }	7e ~	7f del

**FILES**

/usr/pub/ascii

**NAME**

environ — user environment

**DESCRIPTION**

An array of strings called the “environment” is made available by *exec(2)* when a process begins. By convention, these strings have the form “name=value”. The following names are used by various commands:

- PATH** The sequence of directory prefixes that *sh(1)*, *time(1)*, *nice(1)*, *nohup(1)*, etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (:). *Login(1)* sets **PATH**=:/bin:/usr/bin.
- HOME** Name of the user’s login directory, set by *login(1)* from the password file *passwd(4)*.
- TERM** The kind of terminal for which output is to be prepared. This information is used by commands, such as *mm(1)* or *tplot(1G)*, which may exploit special capabilities of that terminal.
- TZ** Time zone information. The format is *xxxnzzz* where *xxx* is standard local time zone abbreviation, *n* is the difference in hours from GMT, and *zzz* is the abbreviation for the daylight-saving local time zone, if any; for example, *EST5EDT*.

Further names may be placed in the environment by the *export* command and “name=value” arguments in *sh(1)*, or by *exec(2)*. It is unwise to conflict with certain shell variables that are frequently exported by *.profile* files: *MAIL*, *PS1*, *PS2*, *IFS*.

**SEE ALSO**

*env(1)*, *login(1)*, *sh(1)*, *exec(2)*, *getenv(3C)*, *profile(4)*, *term(5)*.

## NAME

eqnchar — special character definitions for eqn and neqn

## SYNOPSIS

eqn /usr/pub/eqnchar [ files ] | troff [ options ]

neqn /usr/pub/eqnchar [ files ] | nroff [ options ]

## DESCRIPTION

*Eqnchar* contains *troff*(1) and *nroff* character definitions for constructing characters that are not available on the Wang Laboratories, Inc. C/A/T phototypesetter. These definitions are primarily intended for use with *eqn*(1) and *neqn*; *eqnchar* contains definitions for the following characters:

<i>ciplus</i>	⊕			<i>square</i>	□
<i>citimes</i>	⊗	<i>langle</i>	<	<i>circle</i>	○
<i>wig</i>	~	<i>rangle</i>	>	<i>blot</i>	■
<i>-wig</i>	≈	<i>hbar</i>	ℏ	<i>bullet</i>	●
<i>&gt;wig</i>	≳	<i>ppd</i>	⊥	<i>prop</i>	∝
<i>&lt;wig</i>	≲	<->	↔	<i>empty</i>	∅
<i>=wig</i>	≐	<=>	↔	<i>member</i>	∈
<i>star</i>	*	<	⋈	<i>nomem</i>	ℓ
<i>bigstar</i>	⋆	>	⋈	<i>cup</i>	∪
<i>=dot</i>	≐	<i>ang</i>	∠	<i>cap</i>	∩
<i>orsign</i>	∨	<i>rang</i>	∟	<i>incl</i>	⊆
<i>andsign</i>	∧	<i>3dot</i>	⋮	<i>subset</i>	⊂
<i>=del</i>	≐	<i>thf</i>	∴	<i>supset</i>	⊃
<i>oppA</i>	∇	<i>quarter</i>	¼	<i>!subset</i>	⊈
<i>oppE</i>	≡	<i>3quarter</i>	¾	<i>!supset</i>	⊉
<i>angstrom</i>	Å	<i>degree</i>	°	<i>scrL</i>	ℓ
<i>==&lt;</i>	≐	<i>==&gt;</i>	≐		

## FILES

/usr/pub/eqnchar

## SEE ALSO

eqn(1), nroff(1), troff(1).

**NAME**

fcntl — file control options

**SYNOPSIS**

#include &lt;fcntl.h&gt;

**DESCRIPTION**

The *fcntl(2)* function provides for control over open files. This include file describes *requests* and *arguments* to *fcntl* and *open(2)*.

```
/* Flag values accessible to open(2) and fcntl(2) */
```

```
/* (The first three can only be set by open) */
```

```
#define O_RDONLY 0
```

```
#define O_WRONLY 1
```

```
#define O_RDWR 2
```

```
#define O_NDELAY 04 /* Non-blocking I/O */
```

```
#define O_APPEND 010 /* append (writes guaranteed at the end) */
```

```
/* Flag values accessible only to open(2) */
```

```
#define O_CREAT 00400 /* open with file create (uses third open arg)*/
```

```
#define O_TRUNC 01000 /* open with truncation */
```

```
#define O_EXCL 02000 /* exclusive open */
```

```
/* fcntl(2) requests */
```

```
#define F_DUPFD 0 /* Duplicate files */
```

```
#define F_GETFD 1 /* Get files flags */
```

```
#define F_SETFD 2 /* Set files flags */
```

```
#define F_GETFL 3 /* Get file flags */
```

```
#define F_SETFL 4 /* Set file flags */
```

**SEE ALSO**

fcntl(2), open(2).

**NAME**

greek — graphics for the extended TTY-37 type-box

**SYNOPSIS**

cat /usr/pub/greek [ | greek -Tterminal ]

**DESCRIPTION**

*Greek* gives the mapping from ASCII to the “shift-out” graphics in effect between SO and SI on TELETYPE® Model 37 terminals equipped with a 128-character type-box. These are the default greek characters produced by *nroff*. The filters of *greek*(1) attempt to print them on various other terminals. The file contains:

alpha	$\alpha$	A	beta	$\beta$	B	gamma	$\gamma$	\
GAMMA	$\Gamma$	G	delta	$\delta$	D	DELTA	$\Delta$	W
epsilon	$\epsilon$	S	zeta	$\zeta$	Q	eta	$\eta$	N
THETA	$\Theta$	T	theta	$\theta$	O	lambda	$\lambda$	L
LAMBDA	$\Lambda$	E	mu	$\mu$	M	nu	$\nu$	@
xi	$\xi$	X	pi	$\pi$	J	PI	$\Pi$	P
rho	$\rho$	K	sigma	$\sigma$	Y	SIGMA	$\Sigma$	R
tau	$\tau$	I	phi	$\phi$	U	PHI	$\Phi$	F
psi	$\psi$	V	PSI	$\Psi$	H	omega	$\omega$	C
OMEGA	$\Omega$	Z	nabla	$\nabla$	[	not	$\neg$	-
partial	$\partial$	]	integral	$\int$	^			

**FILES**

/usr/pub/greek

**SEE ALSO**

300(1), 4014(1), 450(1), greek(1), hp(1), tc(1), nroff(1).

## NAME

man — macros for formatting entries in this manual

## SYNOPSIS

**nroff** — man files

**troff** — man [ -rs1 ] files

## DESCRIPTION

These *troff*(1) macros are used to lay out the format of the entries of this manual. A skeleton entry may be found in the file `/usr/man/u_man/man0/skeleton`. These macros are used by the *man*(1) command.

The default page size is 8.5"×11", with a 6.5"×10" text area; the `-rs1` option reduces these dimensions to 6"×9" and 4.75"×8.375", respectively; this option (which is *not* effective in *nroff*) also reduces the default type size from 10-point to 9-point, and the vertical line spacing from 12-point to 10-point. The `-rv2` option may be used to set certain parameters to values appropriate for certain Versatec printers: it sets the line length to 82 characters, the page length to 84 lines, and it inhibits underlining; this option should not be confused with the `-Tvp` option of the *man*(1) command, which is available at some UNIX sites.

Any *text* argument below may be one to six "words". Double quotes (") may be used to include blanks in a "word". If *text* is empty, the special treatment is applied to the next line that contains text to be printed. For example, `.I` may be used to italicize a whole line, or `.SM` followed by `.B` to make small bold text. By default, hyphenation is turned off for *nroff*, but remains on for *troff*.

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros, e.g., `.I`, `.RB`, `.SM`. Tab stops are neither used nor set by any macro except `.DT` and `.TH`.

Default units for indents *in* are ens. When *in* is omitted, the previous indent is used. This remembered indent is set to its default value (7.2 ens in *troff*, 5 ens in *nroff*—this corresponds to 0.5" in the default page size) by `.TH`, `.P`, and `.RS`, and restored by `.RE`.

`.TH t s c n` Set the title and entry heading; *t* is the title, *s* is the section number, *c* is extra commentary, e.g., "local", *n* is new manual name. Invokes `.DT` (see below).

`.SH text` Place subhead *text*, e.g., SYNOPSIS, here.

`.SS text` Place sub-subhead *text*, e.g., Options, here.

`.B text` Make *text* bold.

`.I text` Make *text* italic.

`.SM text` Make *text* 1 point smaller than default point size.

`.RI a b` Concatenate roman *a* with italic *b*, and alternate these two fonts for up to six arguments. Similar macros alternate between any two of roman, italic, and bold:

`.IR .RB .BR .IB .BI`

`.P` Begin a paragraph with normal font, point size, and indent. `.PP` is a synonym for `.P`.

`.HP in` Begin paragraph with hanging indent.

`.TP in` Begin indented paragraph with hanging tag. The next line that contains text to be printed is taken as the tag. If the tag does not fit, it is printed on a separate line.

`.IP t in` Same as `.TP in` with tag *t*; often used to get an indented paragraph without a tag.



- .RS** *in*     Increase relative indent (initially zero). Indent all output an extra *in* units from the current left margin.
- .RE** *k*     Return to the *k*th relative indent level (initially, *k*=1; *k*=0 is equivalent to *k*=1); if *k* is omitted, return to the most recent lower indent level.
- .PM** *m*     Produces proprietary markings; where *m* may be P for PRIVATE, N for NOTICE, BP for BELL LABORATORIES PROPRIETARY, or BR for BELL LABORATORIES RESTRICTED.
- .DT**       Restore default tab settings (every 7.2 ens in *troff*, 5 ens in *nroff*).
- .PD** *v*     Set the interparagraph distance to *v* vertical spaces. If *v* is omitted, set the interparagraph distance to the default value (0.4*v* in *troff*, 1*v* in *nroff*).

The following *strings* are defined:

- \oR**       ◦ in *troff*, (Reg.) in *nroff*.
- \oS**       Change to default type size.
- \\*(Tm**     Trademark indicator.

The following *number registers* are given default values by .TH:

- IN**       Left margin indent relative to subheads (default is 7.2 ens in *troff*, 5 ens in *nroff*).
- LL**       Line length including IN.
- PD**       Current interparagraph distance.

## CAVEATS

In addition to the macros, strings, and number registers mentioned above, there are defined a number of *internal* macros, strings, and number registers. Except for names predefined by *troff* and number registers *d*, *m*, and *y*, all such internal names are of the form *XA*, where *X* is one of ), ], and }, and *A* stands for any alphanumeric character.

If a manual entry needs to be preprocessed by *cw*(1), *eqn*(1) (or *neqn*), and/or *tbl*(1), it must begin with a special line (described in *man*(1)), causing the *man* command to invoke the appropriate preprocessor(s).

The programs that prepare the Table of Contents and the Permuted Index for this Manual assume the *NAME* section of each entry consists of a single line of input that has the following format:

```
name[, name, name ...] \- explanatory text
```

The macro package increases the inter-word spaces (to eliminate ambiguity) in the *SYNOPSIS* section of each entry.

The macro package itself uses only the roman font (so that one can replace, for example, the bold font by the constant-width font—see *cw*(1)). Of course, if the input text of an entry contains requests for other fonts (e.g., .I, .RB, \fI), the corresponding fonts must be mounted.

## FILES

```
/usr/lib/tmac/tmac.an
/usr/lib/macros/cmp.[nt].[dt].an
/usr/lib/macros/ucmp.[nt].an
/usr/man/[ua]_man/man0/skeleton
```

**SEE ALSO**

man(1), nroff(1), troff(1).

**BUGS**

If the argument to .TH contains *any* blanks and is *not* enclosed by double quotes (""), there will be bird-dropping-like things on the output.

**NAME**

**mm** — the MM macro package for formatting documents

**SYNOPSIS**

**mm** [ options ] [ files ]  
**nroff** — **mm** [ options ] [ files ]  
**nroff** — **cm** [ options ] [ files ]  
  
**mmt** [ options ] [ files ]  
**troff** — **mm** [ options ] [ files ]  
**troff** — **cm** [ options ] [ files ]

**DESCRIPTION**

This package provides a formatting capability for a very wide variety of documents. It is the standard package used by the BTL typing pools and documentation centers. The manner in which a document is typed in and edited is essentially independent of whether the document is to be eventually formatted at a terminal or is to be phototypeset. See the references below for further details.

The **-mm** option causes *nroff* and *troff(1)* to use the non-compacted version of the macro package, while the **-cm** option results in the use of the compacted version, thus speeding up the process of loading the macro package.

**FILES**

<code>/usr/lib/tmac/tmac.m</code>	pointer to the non-compacted version of the package
<code>/usr/lib/macros/mm[nt]</code>	non-compacted version of the package
<code>/usr/lib/macros/cmp.[nt].[dt].m</code>	compacted version of the package
<code>/usr/lib/macros/ucmp.[nt].m</code>	initializers for the compacted version of the package

**SEE ALSO**

`mm(1)`, `mmt(1)`, `nroff(1)`, `troff(1)`.  
*MM—Memorandum Macros* by D. W. Smith and J. R. Mashey.  
*Typing Documents with MM* by D. W. Smith and E. M. Piskorik.

**NAME**

**mosd** — the OSDD adapter macro package for formatting documents

**SYNOPSIS**

```

osdd [ options ] [ files ]
mm -mosd [ options ] [ files ]
nroff -mm -mosd [ options ] [ files ]
nroff -cm -mosd [ options ] [ files ]

mmt -mosd [ options ] [ files ]
troff -mm -mosd [ options ] [ files ]
troff -cm -mosd [ options ] [ files ]

```

**DESCRIPTION**

The OSDD adapter macro package is a tool used in conjunction with the MM macro package to prepare Operations Systems Deliverable Documentation. Many of the OSDD Standards are different than the default format provided by MM. The OSDD adapter package sets the appropriate MM options for automatic production of the OSDD Standards. The OSDD adapter package also generates the correct OSDD page headers and footers, heading styles, Table of Contents format, etc.

OSDD document (input) files are prepared with the MM macros. Additional information which must be given at the beginning of the document file is specified by the following string definitions:

```

.ds H1 document-number
.ds H2 section-number
.ds H3 issue-number
.ds H4 date
.ds H5 rating

```

The *document-number* should be of the standard 10 character format. The words "Section" and "Issue" should not be included in the string definitions; they will be supplied automatically when the document is printed. For example:

```

.ds H1 OPA-1P135-01
.ds H2 4
.ds H3 2

```

automatically produces

```

OPA-1P135-01
Section 4
Issue 2

```

as the document page header. Quotation marks are not used in string definitions.

If certain information is not to be included in a page header, then the string is defined as null; e.g.,

```

.ds H2

```

means that there is no *section-number*.

The OSDD Standards require that the *Table of Contents* be numbered beginning with *Page 1*. By default, the first page of text will be numbered *Page 2*. If the *Table of Contents* has more than one page, for example *n*, then either `-rPn+1` must be included as a command line option or `.nr P n` must be included in the document file. For example, if the *Table of Contents* is four pages then use `-rP5` on the command line or `.nr P 4` in the document file.

The OSDD Standards require that certain information such as the document *rating* appear on the *Document Index* or on the *Table of Contents* page if there is no index. By default, it is assumed that an index has been prepared separately. If there is no index, the following must be included in the document file:

```
.nr Di 0
```

This will ensure that the necessary information is included on the *Table of Contents* page.

The OSDD Standards require that all numbered figures be placed at the end of the document. The `.Fg` macro is used to produce full page figures. This macro produces a blank page with the appropriate header, footer, and figure caption. Insertion of the actual figure on the page is a manual operation. The macro usage is

```
.Fg page-count "figure caption"
```

where *page-count* is the number of pages required for a multi-page figure (default 1 page).

Figure captions are produced by the `.Fg` macro using the `.BS/.BE` macros. Thus the `.BS/.BE` macros are also not available for users. The `.Fg` macro cannot be used within the document unless the final `.Fg` in a series of figures is followed by a `.SK` macro to force out the last figure page.

The *Table of Contents* for OSDD documents (see Figure 4 in Section 4.1 of the OSDD Standards) is produced with:

```
.Tc
System Type
System Name
Document Type
.Td
```

The `.Tc/.Td` macros are used instead of the `.TC` macro from MM.

By default, the adapter package causes the NOTICE disclosure statement to be printed. The `.PM` macro may be used to suppress the NOTICE or to replace it with the PRIVATE disclosure statement as follows:

```
.PM      none printed
.PM P    PRIVATE printed
.PM N    NOTICE printed (default)
```

The `.P` macro is used for paragraphs. The `Np` register is set automatically to indicate the paragraph numbering style. It is very important that the `.P` macro be used correctly. All paragraphs (including those immediately following a `.H` macro) must use a `.P` macro. Unless there is a `.P` macro, there will not be a number generated for the paragraph. Similarly, the `.P` macro should not be used for text which is not a paragraph. The `.SP` macro may be appropriate for these cases, e.g., for "paragraphs" within a list item.

The page header format is produced automatically in accordance with the OSDD Standards. The OSDD Adapter macro package uses the `.TP` macro for this purpose. Therefore the `.TP` macro normally available in MM is not available for users.

## FILES

/usr/lib/tmac/tmac.osd

## SEE ALSO

mm(1), mmt(1), nroff(1), troff(1), mm(5).

MM—Memorandum Macros by D. W. Smith and J. R. Mashey.

Operations Systems Deliverable Documentation Standards, June 1980.

**NAME**

mptx — the macro package for formatting a permuted index

**SYNOPSIS**

**nroff** -mptx [ options ] [ files ]

**troff** -mptx [ options ] [ files ]

**DESCRIPTION**

This package provides a definition for the .xx macro used for formatting a permuted index as produced by *ptx*(1). This package does not provide any other formatting capabilities such as headers and footers. If these or other capabilities are required, the *mptx* macro package may be used in conjunction with the *MM* macro package. In this case, the -mptx option must be invoked *after* the -mm call. For example:

nroff -cm -mptx file

or

mm -mptx file

**FILES**

/usr/lib/tmac/tmac.ptx pointer to the non-compacted version of the package

/usr/lib/macros/ptx non-compacted version of the package

**SEE ALSO**

mm(1), nroff(1), ptx(1), troff(1), mm(5).

## NAME

*mv* — a troff macro package for typesetting view graphs and slides

## SYNOPSIS

*mvt* [ -a ] [ options ] [ files ]

*troff* [ -a ] [ -rX1 ] -*mv* [ options ] [ files ]

## DESCRIPTION

This package makes it easy to typeset view graphs and projection slides in a variety of sizes. A few macros (briefly described below) accomplish most of the formatting tasks needed in making transparencies. All of the facilities of *troff*(1), *cw*(1), *eqn*(1), and *tbl*(1) are available for more difficult tasks.

The output can be previewed on most terminals, and, in particular, on the Tektronix 4014, as well as on the Versatec printer. For these two devices, specify the *-rX1* option (this option is automatically specified by the *mvt* command—q.v.—when that command is invoked with the *-T4014* or *-Tvp* options). To preview output on other terminals, specify the *-a* option.

The available macros are:

**.VS** [n] [i] [d] Foil-start macro; foil size is to be 7"×7"; *n* is the foil number, *i* is the foil identification, *d* is the date; the foil-start macro resets all parameters (indent, point size, etc.) to initial default values, except for the values of *i* and *d* arguments inherited from a previous foil-start macro; it also invokes the *.A* macro (see below).

The naming convention for this and the following eight macros is that the first character of the name (V or S) distinguishes between view graphs and slides, respectively, while the second character indicates whether the foil is square (S), small wide (w), small high (h), big wide (W), or big high (H). Slides are "skinnier" than the corresponding view graphs: the ratio of the longer dimension to the shorter one is larger for slides than for view graphs. As a result, slide foils can be used for view graphs, but not vice versa; on the other hand, view graphs can accommodate a bit more text.

**.Vw** [n] [i] [d] Same as *.VS*, except that foil size is 7" wide × 5" high.  
**.Vh** [n] [i] [d] Same as *.VS*, except that foil size is 5"×7".  
**.VW** [n] [i] [d] Same as *.VS*, except that foil size is 7"×5.4".  
**.VH** [n] [i] [d] Same as *.VS*, except that foil size is 7"×9".  
**.Sw** [n] [i] [d] Same as *.VS*, except that foil size is 7"×5".  
**.Sh** [n] [i] [d] Same as *.VS*, except that foil size is 5"×7".  
**.SW** [n] [i] [d] Same as *.VS*, except that foil size is 7"×5.4".  
**.SH** [n] [i] [d] Same as *.VS*, except that foil size is 7"×9".  
**.A** [x] Place text that follows at the first indentation level (left margin); the presence of *x* suppresses the ½ line spacing from the preceding text.  
**.B** [m] [s] Place text that follows at the second indentation level; text is preceded by a mark; *m* is the mark (default is a large bullet); *s* is the increment or decrement to the point size of the mark with respect to the prevailing point size (default is 0); if *s* is 100, it causes the point size of the mark to be the same as that of the default mark.

- .C** [*m*] [*s*] Same as **.B**, but for the third indentation level; default mark is a dash.
- .D** [*m*] [*s*] Same as **.B**, but for the fourth indentation level; default mark is a small bullet.
- .T** *string* *String* is printed as an over-size, centered title.
- .I** [*in*] [*a*] [*x*] Change the current text indent (does not affect titles); *in* is the indent (in inches unless dimensioned, default is 0); if *in* is signed, it is an increment or decrement; the presence of *a* invokes the **.A** macro (see below) and passes *x* (if any) to it.
- .S** [*p*] [*l*] Set the point size and line length; *p* is the point size (default is "previous"); if *p* is 100, the point size reverts to the *initial* default for the current foil-start macro; if *p* is signed, it is an increment or decrement (default is 18 for **.VS**, **.VH**, and **.SH**, and 14 for the other foil-start macros); *l* is the line length (in inches unless dimensioned; default is 4.2" for **.Vh**, 3.8" for **.Sh**, 5" for **.SH**, and 6" for the other foil-start macros).
- .DF** *n f* [*n f*...] Define font positions; may not appear within a foil's input text (i.e., it may only appear after all the input text for a foil, but before the next foil-start macro); *n* is the position of font *f*; up to four "*n f*" pairs may be specified; the first font named becomes the *prevailing* font; the initial setting is (H is a synonym for G):  
**.DF 1 H 2 I 3 B 4 S**
- .DV** [*a*] [*b*] [*c*] [*d*] Alter the vertical spacing between indentation levels; *a* is the spacing for **.A**, *b* is for **.B**, *c* is for **.C**, and *d* is for **.D**; all non-null arguments must be dimensioned; null arguments leave the corresponding spacing unaffected; initial setting is:  
**.DV .5v .5v .5v 0v**
- .U** *str1* [*str2*] Underline *str1* and concatenate *str2* (if any) to it.

The last four macros in the above list do not cause a break; the **.I** macro causes a break only if it is invoked with more than one argument; all the other macros cause a break.

The macro package also recognizes the following upper-case synonyms for the corresponding lower-case *troff* requests:

**.AD .BR .CE .FI .HY .NA .NF .NH .NX .SO .SP .TA .TI**

The **Tm** string produces the trademark symbol.

The input tilde (~) character is translated into a blank on output.

See the user's manual cited below for further details.

#### FILES

/usr/lib/tmac/tmac.v  
 /usr/lib/macros/vmca

#### SEE ALSO

**cw(1)**, **eqn(1)**, **mmt(1)**, **tbl(1)**, **troff(1)**.

*A Macro Package for View Graphs and Slides* by T. A. Dolotta and D. W. Smith.

#### BUGS

The **.VW** and **.SW** foils are meant to be 9" wide by 7" high, but because the typesetter paper is generally only 8" wide, they are printed 7" wide by 5.4" high and have to be enlarged by a factor of 9/7 before use as view graphs; this makes them less than totally useful.



## NAME

regexp — regular expression compile and match routines

## SYNOPSIS

```
#define INIT <declarations>
#define GETC() <getc code>
#define PEEKC() <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>

#include <regexp.h>

char *compile(instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;

int step(string, expbuf)
char *string, *expbuf;
```

## DESCRIPTION

This page describes general purpose regular expression matching routines in the form of *ed(1)*, defined in `/usr/include/regexp.h`. Programs such as *ed(1)*, *sed(1)*, *grep(1)*, *bs(1)*, *expr(1)*, etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is unpleasantly complex. Programs that include this file must have the following five macros declared before the “`#include <regexp.h>`” statement. These macros are used by the *compile* routine.

GETC()	Return the value of the next character in the regular expression pattern. Successive calls to GETC() should return successive characters of the regular expression.
PEEKC()	Return the next character in the regular expression. Successive calls to PEEKC() should return the same character (which should also be the next character returned by GETC()).
UNGETC( <i>c</i> )	Cause the argument <i>c</i> to be returned by the next call to GETC() (and PEEKC()). No more than one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC(). The value of the macro UNGETC( <i>c</i> ) is always ignored.
RETURN( <i>pointer</i> )	This macro is used on normal exit of the <i>compile</i> routine. The value of the argument <i>pointer</i> is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.
ERROR( <i>val</i> )	This is the abnormal return from the <i>compile</i> routine. The argument <i>val</i> is an error number (see table below for meanings). This call should never return.

ERROR	MEANING
11	Range endpoint too large.
16	Bad number.
25	"\digit" out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	\( \) imbalance.
43	Too many \(.
44	More than 2 numbers given in \{ \}.
45	} expected after \.
46	First number exceeds second in \{ \}.
49	[ ] imbalance.
50	Regular expression overflow.

The syntax of the *compile* routine is as follows:

```
compile(instring, expbuf, endbuf, eof)
```

The first parameter *instring* is never used explicitly by the *compile* routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char \*) 0) for this parameter.

The next parameter *expbuf* is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (*endbuf* - *expbuf*) bytes, a call to ERROR(50) is made.

The parameter *eof* is the character which marks the end of the regular expression. For example, in *ed*(1), this character is usually a /.

Each program that includes this file must have a *#define* statement for INIT. This definition will be placed right after the declaration for the function *compile* and the opening curly brace ({}). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for GETC(), PEEKC() and UNGETC(). Otherwise it can be used to declare external variables that might be used by GETC(), PEEKC() and UNGETC(). See the example below of the declarations taken from *grep*(1).

There are other functions in this file which perform actual regular expression matching, one of which is the function *step*. The call to *step* is as follows:

```
step(string, expbuf)
```

The first parameter to *step* is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter *expbuf* is the compiled regular expression which was obtained by a call of the function *compile*.

The function *step* returns one, if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to *step*. The variable set in *step* is *loc1*. This is a pointer to the first character that matched the regular expression. The variable *loc2*, which is set by the function *advance*, points the character after the last character that matches the regular expression. Thus if the regular expression matches the entire

line, *loc1* will point to the first character of *string* and *loc2* will point to the null at the end of *string*.

*Step* uses the external variable *circf* which is set by *compile* if the regular expression begins with `^`. If this is set then *step* will only try to match the regular expression to the beginning of the string. If more than one regular expression is to be compiled before the first is executed the value of *circf* should be saved for each compiled expression and *circf* should be set to that saved value before each call to *step*.

The function *advance* is called from *step* with the same arguments as *step*. The purpose of *step* is to step through the *string* argument and call *advance* until *advance* returns a one indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, *step* need not be called, simply call *advance*.

When *advance* encounters a `*` or `\{ \}` sequence in the regular expression it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, *advance* will back up along the string until it finds a match or reaches the point in the string that initially matched the `*` or `\{ \}`. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer *locs* is equal to the point in the string at sometime during the backing up process, *advance* will break out of the loop that backs up and will return zero. This is used by *ed(1)* and *sed(1)* for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like `s/y*/g` do not loop forever.

The routines *ecmp* and *getrange* are trivial and are called by the routines previously mentioned.

## EXAMPLES

The following is an example of how the regular expression macros and calls look from *grep(1)*:

```
#define INIT          register char *sp = instring;
#define GETC()        (*sp++)
#define PEEKC()       (*sp)
#define UNGETC(c)    (--sp)
#define RETURN(c)    return;
#define ERROR(c)     regerr()

#include <regexp.h>
...
                compile(*argv, expbuf, &expbuf[ESIZE], ^0');
...
                if(step(linebuf, expbuf)
                    succeed());
```

## FILES

`/usr/include/regexp.h`

## SEE ALSO

*ed(1)*, *grep(1)*, *sed(1)*.

## BUGS

The handling of *circf* is kludgy.

The routine *ecmp* is equivalent to the Standard I/O routine *strncmp* and should be replaced by that routine.

The actual code is probably easier to understand than this manual page.

## NAME

stat — data returned by stat system call

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
```

## DESCRIPTION

The system calls *stat* and *fstat* return data whose structure is defined by this include file. The encoding of the field *st\_mode* is defined in this file also.

```
/*
 * Structure of the result of stat
 */
```

```
struct  stat
{
    dev_t    st_dev;
    ino_t    st_ino;
    ushort   st_mode;
    short    st_nlink;
    ushort   st_uid;
    ushort   st_gid;
    dev_t    st_rdev;
    off_t    st_size;
    time_t   st_atime;
    time_t   st_mtime;
    time_t   st_ctime;
};
```

```
#define S_IFMT    0170000 /* type of file */
#define S_IFDIR   0040000 /* directory */
#define S_IFCHR   0020000 /* character special */
#define S_IFBLK   0060000 /* block special */
#define S_IFREG   0100000 /* regular */
#define S_IFIFO   0010000 /* fifo */
#define S_ISUID   04000   /* set user id on execution */
#define S_ISGID   02000   /* set group id on execution */
#define S_ISVTX   01000   /* save swapped text even after use */
#define S_IRREAD  00400   /* read permission, owner */
#define S_IWWRITE 00200   /* write permission, owner */
#define S_IXEXEC  00100   /* execute/search permission, owner */
```

## FILES

```
/usr/include/sys/types.h
/usr/include/sys/stat.h
```

## SEE ALSO

stat(2), types(5).

## NAME

term — conventional names for terminals

## DESCRIPTION

These names are used by certain commands (e.g., *nroff*, *mm*(1), *man*(1), *tabs*(1)) and are maintained as part of the shell environment (see *sh*(1), *profile*(4), and *environ*(5)) in the variable **STERM**:

1520	Datamedia 1520
1620	Diablo 1620 and others using the HyType II printer
1620-12	same, in 12-pitch mode
2621	Hewlett-Packard HP2621 series
2631	Hewlett-Packard 2631 line printer
2631-c	Hewlett-Packard 2631 line printer - compressed mode
2631-e	Hewlett-Packard 2631 line printer - expanded mode
2640	Hewlett-Packard HP2640 series
2645	Hewlett-Packard HP264n series (other than the 2640 series)
300	DASI/DTC/GSI 300 and others using the HyType I printer
300-12	same, in 12-pitch mode
300s	DASI/DTC/GSI 300s
382	DTC 382
300s-12	same, in 12-pitch mode
3045	Datamedia 3045
33	TELETYPE® Model 33 KSR
37	TELETYPE Model 37 KSR
40-2	TELETYPE Model 40/2
40-4	TELETYPE Model 40/4
4540	TELETYPE Model 4540
3270	IBM Model 3270
4000a	Trendata 4000a
4014	Tektronix 4014
43	TELETYPE Model 43 KSR
450	DASI 450 (same as Diablo 1620)
450-12	same, in 12-pitch mode
735	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
sync	generic name for synchronous TELETYPE 4540-compatible terminals
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
tn1200	General Electric TermiNet 1200
tn300	General Electric TermiNet 300

Up to 8 characters, chosen from [-a-z0-9], make up a basic terminal name. Terminal sub-models and operational modes are distinguished by suffixes beginning with a -. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name.

Commands whose behavior depends on the type of terminal should accept arguments of the form *-Term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **STERM**, which, in turn, should contain *term*.

## SEE ALSO

*mm*(1), *nroff*(1), *tplot*(1G), *sh*(1), *stty*(1), *tabs*(1), *profile*(4), *environ*(5).

**BUGS**

This is a small candle trying to illuminate a large, dark problem. Programs that ought to adhere to this nomenclature do so somewhat fitfully.

## NAME

types — primitive system data types

## SYNOPSIS

```
#include <sys/types.h>
```

## DESCRIPTION

The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```
typedef struct { int r[1]; } *      physadr;
typedef long      daddr_t;
typedef char *    caddr_t;
typedef unsigned int  uint;
typedef unsigned short ushort;
typedef ushort    ino_t;
typedef short     cnt_t;
typedef long      time_t;
typedef int       label_t[10];
typedef short     dev_t;
typedef long      off_t;
typedef long      paddr_t;
typedef long      key_t;
```

The form *daddr\_t* is used for disk addresses except in an i-node on disk, see *fs(4)*. Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The *label\_t* variables are used to save the processor state while another process is running.

## SEE ALSO

*fs(4)*.

**NAME**

intro — introduction to games

**DESCRIPTION**

This section describes the recreational and educational programs found in the directory `/usr/games`. The availability of these programs may vary from system to system.



**NAME**

arithmetic -- provide drill in number facts

**SYNOPSIS**

/usr/games/arithmetic [ +-x/ ] [ range ]

**DESCRIPTION**

*Arithmetic* types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; +, -, x, and / respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is +-.

*Range* is a decimal number; all addends, subtrahends, differences, multipliers, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

**NAME**

back — the game of backgammon

**SYNOPSIS**`/usr/games/back`**DESCRIPTION**

*Back* is a program which provides a partner for the game of backgammon. It is designed to play at three different levels of skill, one of which you must select. In addition to selecting the opponent's level, you may also indicate that you would like to roll your own dice during your turns (for the superstitious players). You will also be given the opportunity to move first. The practice of each player rolling one die for the first move is not incorporated.

The points are numbered 1–24, with 1 being white's extreme inner table, 24 being brown's inner table, 0 being the bar for removed white pieces and 25 the bar for brown. For details on how moves are expressed, type `y` when *back* asks "Instructions?" at the beginning of the game. When *back* first asks "Move?", type `?` to see a list of move options other than entering your numerical move.

When the game is finished, *back* will ask you if you want the log. If you respond with `y`, *back* will attempt to append to or create a file `back.log` in the current directory.

**FILES**

<code>/usr/games/lib/backrules</code>	rules file
<code>/tmp/b*</code>	log temp file
<code>back.log</code>	log file

**BUGS**

The only level really worth playing is "expert", and it only plays the forward game.

*Back* will complain loudly if you attempt to make too *many* moves in a turn, but will become very silent if you make too *few*.

Doubling is not implemented.

## NAME

bj — the game of black jack

## SYNOPSIS

/usr/games/bj

## DESCRIPTION

*Bj* is a serious attempt at simulating the dealer in the game of black jack (or twenty-one) as might be found in Reno. The following rules apply:

The bet is \$2 every hand.

A player "natural" (black jack) pays \$3. A dealer natural loses \$2. Both dealer and player naturals is a "push" (no money exchange).

If the dealer has an ace up, the player is allowed to make an "insurance" bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where the player wins \$2 if the dealer has a natural and loses \$1 if the dealer does not.

If the player is dealt two cards of the same value, he is allowed to "double". He is allowed to play two hands, each with one of these cards. (The bet is doubled also; \$2 on each hand.)

If a dealt hand has a total of ten or eleven, the player may "double down". He may double the bet (\$2 to \$4) and receive exactly one more card on that hand.

Under normal play, the player may "hit" (draw a card) as long as his total is not over twenty-one. If the player "busts" (goes over twenty-one), the dealer wins the bet.

When the player "stands" (decides not to hit), the dealer hits until he attains a total of seventeen or more. If the dealer busts, the player wins the bet.

If both player and dealer stand, the one with the largest total wins. A tie is a push.

The machine deals and keeps score. The following questions will be asked at appropriate times. Each question is answered by y followed by a new-line for "yes", or just new-line for "no".

? (means, "do you want a hit?")

Insurance?

Double down?

Every time the deck is shuffled, the dealer so states and the "action" (total bet) and "standing" (total won or lost) is printed. To exit, hit the interrupt key (DEL) and the action and standing will be printed.

**NAME**

chess — the game of chess

**SYNOPSIS**

/usr/games/chess

**DESCRIPTION**

*Chess* is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol + must be placed at the end of a line when the move on that line places the opponent's king in check. o-o and o-o-o specify castling, king side or queen side, respectively.

The user is prompted for a move or command by a \*. To play black, type first at the onset of the game. To print a copy of the board in play, type a carriage return only. Each move is echoed in the appropriate notation, followed by the program's reply. Near the middle and end games, the program can take considerable time in computing its moves.

A ? or help may be typed to get a help message that briefly describes the possible commands.

**DIAGNOSTICS**

The most cryptic diagnostic is "eh?" which means that the input was syntactically incorrect.

**BUGS**

Pawns may be promoted only to queens.

## NAME

craps — the game of craps

## SYNOPSIS

/usr/games/craps

## DESCRIPTION

*Craps* is a form of the game of craps that is played in Las Vegas. The program simulates the *roller*, while the user (the *player*) places bets. The player may choose, at any time, to bet with the roller or with the *House*. A bet of a negative amount is taken as a bet with the House, any other bet is a bet with the roller.

The player starts off with a "bankroll" of \$2,000.

The program prompts with:

bet?

The bet can be all or part of the player's bankroll. Any bet over the total bankroll is rejected and the program prompts with bet? until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (the player wins or loses depending on whether the bet is placed with the roller or with the House; the odds are even). The *first* roll is the roll immediately following a bet:

1. On the first roll:

7 or 11	wins for the roller;
2, 3, or 12	wins for the House;
any other number	is the <i>point</i> , roll again (Rule 2 applies).

2. On subsequent rolls:

point	roller wins;
7	House wins;
any other number	roll again.

If a player loses the entire bankroll, the House will offer to lend the player an additional \$2,000. The program will prompt:

marker?

A **yes** (or **y**) consummates the loan. Any other reply terminates the game.

If a player owes the House money, the House reminds the player, before a bet is placed, how many markers are outstanding.

If, at any time, the bankroll of a player who has outstanding markers exceeds \$2,000, the House asks:

Repay marker?

A reply of **yes** (or **y**) indicates the player's willingness to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker are outstanding, the House asks:

How many?

markers the player would like to repay. If an invalid number is entered (or just a carriage return), an appropriate message is printed and the program will prompt with **How many?** until a valid number is entered.

If a player accumulates 10 markers (a total of \$20,000 borrowed from the House), the program informs the player of the situation and exits.

Should the bankroll of a player who has outstanding markers exceed \$50,000, the *total* amount of money borrowed will be *automatically* repaid

to the House.

Any player who accumulates \$100,000 or more breaks the bank. The program then prompts:

New game?

to give the House a chance to win back its money.

Any reply other than yes is considered to be a no (except in the case of bet? or How many?). To exit, send an interrupt (break), DEL, or control-D. The program will indicate whether the player won, lost, or broke even.

#### MISCELLANEOUS

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

**NAME**

hangman — guess the word

**SYNOPSIS**

*/usr/games/hangman* [ *arg* ]

**DESCRIPTION**

*Hangman* chooses a word at least seven letters long from a dictionary. The user is to guess letters one at a time.

The optional argument *arg* names an alternate dictionary.

**FILES**

*/usr/lib/w2006*

**BUGS**

Hyphenated compounds are run together.

**NAME**

jotto — secret word game

**SYNOPSIS**

/usr/games/jotto [-p]

**DESCRIPTION**

*Jotto* is a word guessing game. You try to guess the computer's secret word before it guesses yours. Clues are obtained by entering probe words. For example, if the computer's secret word is "brown" and you probe with "stare", it will reply "1" indicating that there is one letter in common between your probe and the secret word. Double letters count only once unless they appear in both words. For example, if the hidden word is "igloo" and you probe with "broke", the computer will reply "1". But if you probe with "gloom", the computer will respond "4". All secret words and probe words should be non-proper English five-letter words. If the computer guesses your word exactly, please respond with "y". It will then tell you what its secret word was. The -p flag instructs the computer to report its progress in guessing your word.

**BUGS**

The dictionary contains some unusual words and lacks some common ones.



**NAME**

maze — generate a maze

**SYNOPSIS**

*/usr/games/maze*

**DESCRIPTION**

*Maze* asks a few questions and then prints a maze.

**BUGS**

Some mazes (especially small ones) have no solutions.

**NAME**

moo — guessing game

**SYNOPSIS**

`/usr/games/moo`

**DESCRIPTION**

*Moo* is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. The player guesses four distinct digits being scored on each guess. A “cow” is a correct digit in an incorrect position. A “bull” is a correct digit in a correct position. The game continues until the player guesses the number (a score of four bulls).

**NAME**

quiz - test your knowledge

**SYNOPSIS**

```
/usr/games/quiz [ -i file ] [ -t ] [ category1 category2 ]
```

**DESCRIPTION**

*Quiz* gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*, or vice versa. If no categories are specified, *quiz* gives instructions and lists the available categories.

*Quiz* tells a correct answer whenever you type a bare new-line. At the end of input, upon interrupt, or when questions run out, *quiz* reports a score and terminates.

The **-t** flag specifies "tutorial" mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **-i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      = category new-line | category : line
category  = alternate | category | alternate
alternate = empty | alternate primary
primary   = character | [ category ] | option
option    = { category }
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash \ is used as with *sh*(1) to quote syntactically significant characters or to insert transparent new-lines into a line. When either a question or its answer is empty, *quiz* will refrain from asking it.

**FILES**

```
/usr/games/lib/quiz/index
/usr/games/lib/quiz/*
```

**BUGS**

The construct "a|ab" doesn't work in an information file. Use "a{b}".

## NAME

reversi — a game of dramatic reversals

## SYNOPSIS

`/usr/games/reversi [ [ -r ] file ]`

## DESCRIPTION

*Reversi* (also known as “friends”, “Chinese friends” and “Othello”) is played on an 8 by 8 board using two-sided tokens. Each player takes his turn by placing a token with his side up in an empty square. During the first four turns, players may only place tokens in the four central squares of the board. Subsequently, with each turn, a player *must* capture one or more of his opponent’s tokens. He does this by placing one of his tokens such that it and another of his tokens embrace a solid line of his opponent’s horizontally, vertically or diagonally. Captured tokens are flipped over and thus can be re-captured. If a player cannot outflank his opponent he forfeits his turn. The play continues until the board is filled or until no more outflanking is possible.

In this game, your tokens are asterisks (\*) and the machine’s are at-signs (@). You move by typing in the row and column at which you want to place your token as two digits (1-8), optionally separated by blanks or tabs.

You can also type in:

- c** to continue the game after hitting break (this is only necessary if you interrupt the machine while it is deliberating),
- g n** to start *reversi* playing against itself for the next *n* moves (or until the break key is hit),
- n** to stop printing the board after each move,
- o** to start it up again,
- p** to print the board regardless,
- q** to quit (without dishonor),
- s** to print the score, and, as always,
- !** to escape to the shell. Control-d gets you back.

*Reversi* also recognizes several commands which are valid only at the start of the game, before any moves have been made. They are:

- f** to let the machine go first.
- h n** to ask for a handicap of from one to four corner squares. If you’re *really* good, you can give the machine a handicap by typing a negative number.
- l n** to set the amount of look-ahead used by the machine in searching for moves. Zero means none at all. Four is the default. Greater than six means you may fall asleep waiting for the machine to move.
- t n** to tell *reversi* that you will only need *n* seconds to consider each move. If you fail to respond in the allotted time, you forfeit your turn.

If *reversi* is given a file name as an argument, it will checkpoint the game, move by move, by dumping the board onto *file*. The `-r` option will cause *reversi* to restart the game from *file* and continue logging.

## DIAGNOSTICS

“Illegal!” for an illegal move, and “Huh?” for a move that even the machine cannot understand.

**NAME**

sky — obtain ephemerides

**SYNOPSIS**

/usr/games/sky [ -1 ]

**DESCRIPTION**

*Sky* predicts the apparent locations of the Sun, the Moon, the planets out to Saturn, stars of magnitude at least 2.5, and certain other celestial objects. *Sky* reads the standard input to obtain a GMT time typed on one line with blanks separating year, month number, day, hour, and minute; if the year is missing the current year is used. If a blank line is typed the current time is used. The program prints the azimuth, elevation, and magnitude of objects which are above the horizon at the ephemeris location of Murray Hill at the indicated time. The -1 flag causes it to ask for another location.

Placing a "1" input after the minute entry causes the program to print out the Greenwich Sidereal Time at the indicated moment and to print for each body its topographic right ascension and declination as well as its azimuth and elevation. Also, instead of the magnitude, the semidiameter of the body, in seconds of arc, is reported.

A "2" after the minute entry makes the coordinate system geocentric.

The effects of atmospheric extinction on magnitudes are not included; the brightest magnitudes of variable stars are marked with \*.

For all bodies, the program takes into account precession and nutation of the equinox, annual (but not diurnal) aberration, diurnal parallax, and the proper motion of stars. In no case is refraction included.

The program takes into account perturbations of the Earth due to the Moon, Venus, Mars, and Jupiter. The expected accuracies are: for the Sun and other stellar bodies a few tenths of seconds of arc; for the Moon (on which particular care is lavished) likewise a few tenths of seconds. For the Sun, Moon and stars the accuracy is sufficient to predict the circumstances of eclipses and occultations to within a few seconds of time. The planets may be off by several minutes of arc.

There are lots of special options not described here, which do things like substituting named star catalogs, smoothing nutation and aberration to aid generation of mean places of stars, and making conventional adjustments to the Moon to improve eclipse predictions.

For the most accurate use of the program it is necessary to know that it actually runs in Ephemeris time.

**SEE ALSO**

*American Ephemeris and Nautical Almanac*, for the appropriate years; also, *the Explanatory Supplement to the American Ephemeris and Nautical Almanac*.

**NAME**

ttt, cubic — tic-tac-toe

**SYNOPSIS**

*/usr/games/ttt*  
*/usr/games/cubic*

**DESCRIPTION**

*Ttt* is the X and O game popular in the first grade. This is a learning program that never makes the same mistake twice.

Although it learns, it learns slowly. It must lose nearly 80 games to completely know the game.

*Cubic* plays three-dimensional tic-tac-toe on a 4×4×4 board. Moves are specified as a sequence of three coordinate numbers in the range 1-4.

**FILES**

*/usr/games/ttt.k*      learning file

**BUGS**

*Cubic does not yet work on VAX.*

**NAME**

wump — the game of hunt-the-wumpus

**SYNOPSIS**

`/usr/games/wump`

**DESCRIPTION**

*Wump* plays the game of "Hunt the Wumpus." A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company*, 2, 2 (November 1973).

**BUGS**

It will never replace Adventure.

