AT&T

386 UNIX® System V Release 3.1

User/System Administrator's Reference Manual

AT&T Products and Services

To order documents from the Customer Information Center:

- Within the continental United States, call 1-800-432-6600
- Outside the continental United States, call 1-317-352-8556
- Send mail orders to:
  AT&T Customer Information Center
  Customer Service Representative
  P.O. Box 19901
  Indianapolis, Indiana 46219

To sign up for UNIX system or AT&T computer courses:

- Within the continental United States, call 1-800-221-1647
- Outside the continental United States, call 1-609-639-4458

To contact marketing representatives about AT&T computer hardware products and UNIX software products:

- Within the continental United States, call 1-800-372-2447
- Outside the continental United States, call collect 1-215-266-2973 or 1-215-266-2975

To find out about UNIX system source licenses:

- Within the continental United States, except North Carolina, call 1-800-828-UNIX

- In North Carolina and outside the continental United States, call 1-919-279-3666

- Or write to:

  Software Licensing
  Guilford Center
  P.O. Box 25000
  Greensboro, NC 27420

Introduction

Introduction

Introduction

This manual describes the features of the UNIX system. It provides neither a general overview of the UNIX system nor details of the implementation of the system. Commands that constitute the basic software running on your computer are described.

The manual is divided into four sections:

- 1 - System Commands, System Maintenance Commands, and Application Programs
- 7 - Special Files
- 8 - System Maintenance Procedures
- Index to Packages.

Throughout this volume each reference of the form *name*(1M), *name*(7), *name*(8), or followed by a (1), (1C), or (1G), refers to entries in this manual. The numbers following the command are intended for easy cross-reference. (Section 1 commands appropriate for use by programmers are located in the *Programmer's Reference Manual.*) All other references to entries of the form *name*(N), where *N* is a number [(2), (3), (4), or (5)] possibly followed by a letter, refer to entry *name* in Section *N* of the *Programmer's Reference Manual.*

Each entry in the "Commands" section appears under a single name shown at the upper corners of its page(s). Entries are alphabetized, with the exception of the *intro*(1) entry, which is first. Entries may consist of more than one page. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "primary" name, the name that appears at the upper corners of the page. An example of such an entry is **mount**(1M), which also describes the **umount** command. The "secondary" commands are listed directly below their associated primary command.

Section 1 (*System Commands, System Maintenance Commands and Application Programs*) contains commands and programs that are:

- Used in administering a UNIX system.
- Invoked directly by the user or by command language procedures, as opposed to subroutines, which are called by the user's programs.

Commands generally reside in the directory **/bin** (for **bin**ary programs). In addition, some programs reside in **/usr/bin**. These directories are searched automatically by the command interpreter called the *shell*. The *shell* will search the path in your **.profile**. Make sure you have set this path in your **.profile** file. UNIX systems running on your computer also have a directory called **/usr/lbin**, containing local commands.

The following sub-classes are in this section:

- 1 – General-purpose Commands
- 1C – Communications Commands
- 1G – Graphics Commands
- 1M – Maintenance Commands

Each entry in the "Commands" section appears under a single name shown at the upper corners of its page(s).

Section 7 (*Special Files*) discusses the characteristics of system files that refer to input/output devices. The names in this section generally refer to device names for the hardware, rather than to the names of the special files themselves.

Section 8 (*System Maintenance Procedures*) discusses crash recovery, firmware programs, boot procedures, facility descriptions, etc.

Index to Packages. The utilities packages represented in this section are:

1. Base System
2. Editing Package
3. Remote Terminal Package

The Security Administration Utilities Package is expressly provided for U. S. customers.

All entries are presented using the following format (though some of these headings might not appear in every entry):

- **NAME** gives the primary name [and secondary name(s), as the case may be] and briefly states its purpose.
- **SYNOPSIS** summarizes the usage of the program being described. A few explanatory conventions are used, particularly in Section 1M and the **SYNOPSIS**:

  - **Boldface** strings are literals and are to be typed just as they appear.
  - *Italic* strings usually represent substitutable argument prototypes and command 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 an 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 =.

- **DESCRIPTION** discusses how to use these commands.
- **EXAMPLE(S)** gives example(s) of usage, where appropriate.
- **FILES** contains the file names that are referenced by the program.
- **EXIT CODES** discusses values set when the command terminates. The value set is available in the shell environment variable '?' [see sh(1)].
- **NOTES** gives information that may be helpful under the particular circumstances described.
- **SEE ALSO** offers pointers to related information.
- **DIAGNOSTICS** discusses the error messages that may be produced. Messages that are intended to be self-explanatory are not listed.
- **WARNINGS** discusses the limits or boundaries of the respective commands.
- **BUGS** lists known faults in software that have not been rectified. Occasionally, a suggested short-term remedy is also described.

Preceding Section 1 are a "Table of Contents" (listing both primary and secondary command entries) and a "Permuted Index." Each line of the "Table of Contents" lists an abstract of the command. The "Permuted Index" is used by searching the middle column for a key word or phrase. The right column will then contain the name of the manual page that contains the command. The left column contains additional useful information about the command.

How to Get Started

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

Logging In

You must connect to the UNIX system from the console or a full-duplex ASCII terminal. You must also have a valid login id, which may be obtained [together with how to access your UNIX system] from the administrator of your system. Common terminal speeds are 120, 240, 480, and 960 characters per second (1200, 2400, 4800, and 9600 baud). Some UNIX systems have different ways of accessing each available terminal speed, while other systems offer several speeds through a common access method. In the latter case, there is one "preferred" speed; if you access it 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," "interrupt," or "attention" key until the **login:** message appears.

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 has been established, the system types **login:**. You respond by typing your login id followed by the "return" key. If you have a password, the system asks for it but will not print, or "echo," it on the terminal. After you have logged in, the "return," "new-line," and "line-feed" keys all have equivalent meanings.

Make sure you type your login name in lowercase letters. Typing uppercase letters causes the UNIX system to assume that your terminal can generate only uppercase letters and will treat all letters as uppercase for the remainder of your login session.

When you log in, a message-of-the-day may greet you before you receive your prompt. For more information, consult *login*(1), which discusses the login sequence in more detail, and *stty*(1), which tells you how to describe your terminal to the system. *profile*(4) (in the *Programmer's Reference Manual*) explains how to accomplish this last task automatically every time you log in.

Logging Out

There are two ways to log out:

- If you've dialed in, you can simply hang up the phone.
- 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 the UNIX system, your individual characters are being gathered and temporarily saved. Although they are echoed back to you, these characters will not be given to a program until you type a "return" (or "new-line") as described above in "Logging In."

UNIX system 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, your input characters will have output characters interspersed among them. In any case, 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.

The character **@** cancels all the characters typed before it on a line, effectively deleting the line. (**@** is called the line kill character.) 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 line kill characters can be changed; see *stty*(1).

CONTROL-S (also known as the ASCII **DC3** character) is typed by pressing the control key and the alphabetic **s** simultaneously and is used to stop output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. Output is resumed when a **CONTROL-Q** (also known as **DC1**) is typed. Thus, if you had typed **cat yourfile** and the contents of **yourfile** were passing by on the screen more rapidly than you could read it, you would type **CONTROL-S** to freeze the output for a moment. Typing **CONTROL-Q** would allow the output to resume its rapid pace. The **CONTROL-S** and **CONTROL-Q** 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 do not want. Programs, however, can arrange either to ignore this signal altogether or to be notified and take a specific action 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 an interrupt can be used to halt an editor printout without losing the file being edited.

Besides adapting to the speed of the terminal, the UNIX system 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.

Tab characters are used freely in UNIX system 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 the UNIX system, a program called the shell is communicating with your terminal. The shell reads each line you type, splits the line 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, such as **/bin** and **/usr/bin**. 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 instruct the shell to find them there. See the manual entry for *sh*(1), under the sub-heading "Parameter Substitution," for the discussion of the **$PATH** shell environment variable.

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 or tab characters.

When a program terminates, the shell will ordinarily regain control and give you back your prompt 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

The UNIX system has a file system arranged in a hierarchy of directories. When you received your login id, the system administrator also created a directory for you (ordinarily with the same name as your login id, 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 remove its contents. Permissions to enter or modify 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 or directories 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 or directory name is reached (e.g., **/usr/ae/filex** refers to file **filex** in directory **ae**, while **ae** is itself a subdirectory of **usr**, and **usr** is a subdirectory of the root directory). Use *pwd*(1) to print the full path name of the directory you are working in. See *intro*(2) in the *Programmer's Reference Manual* for a formal definition of *path name*.

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

Important commands that affect files are *cp*(1), *mv* (see *cp*(1)), 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* (see *rm*(1)) for removing them.

Text Entry and Display

Almost all text is entered through an editor. Common examples of UNIX system editors are *ed*(1) and *vi*(1). The commands most often used to print text on a terminal are *cat*(1), *pr*(1), and *pg*(1). The *cat*(1) command displays the contents of ASCII text files 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. The *pg*(1) command displays text in successive portions no larger than your terminal screen.

Communicating with Others

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 try to contact you. *mail*(1) or *mailx*(1) will leave a message whose presence will be announced to another user when he or she next logs in and at periodic intervals during the session. To communicate with another user currently logged in, *write*(1) is used. The corresponding entries in this manual also suggest how to respond to these two commands if you are their target.

See the tutorials in Chapter 8 of the *Operations/System Administration Guide* for more information on communicating with others.

Table of Contents

TABLE OF CONTENTS

Permuted Index

PERMUTED INDEX

functions of DASI 300 and/	300, 300s: handle special	300(1)
/special functions of DASI	300 and 300s terminals.	300(1)
of DASI 300 and 300s/ 300,	300s: handle special functions	300(1)
functions of DASI 300 and	300s terminals. /special	300(1)
comparison. diff3:	3-way differential file	diff3(1)
TEKTRONIX 4014 terminal.	4014: paginator for the	4014(1)
paginator for the TEKTRONIX	4014 terminal. 4014:	4014(1)
of the DASI 450 terminal.	450: handle special functions	450(1)
special functions of the DASI	450 terminal. 450: handle	450(1)
object downloader for the	5620 DMD terminal. wtinit:	wtinit(1M)
prevent LP requests.	accept, reject: allow or	accept(1M)
a directory for remote	access. adv: advertise	adv(1M)
of a file. touch: update	access and modification times	touch(1)
disk: random	access bulk storage medium.	disk(7)
fusage: disk	access profiler.	fusage(1M)
copy file systems for optimal	access time. dcopy:	dcopy(1M)
acctcon2: connect-time	accounting. /acctcon1,	acctcon(1M)
acctprc1, acctprc2: process	accounting. acctprc:	acctprc(1M)
turnacct: shell procedures for	accounting. /startup,	acctsh(1M)
/accton, acctwtmp: overview of	accounting and miscellaneous/	acct(1M)
accounting and miscellaneous	accounting commands. /of	acct(1M)
diskusg: generate disk	accounting data by user ID.	diskusg(1M)
search and print process	accounting file(s). acctcom:	acctcom(1)
acctmerg: merge or add total	accounting files.	acctmerg(1M)
summary from per-process	accounting records. /command	acctcms(1M)
wtmpfix: manipulate connect	accounting records. fwtmp,	fwtmp(1M)
runacct: run daily	accounting.	runacct(1M)
accton, acctwtmp: overview of/	acct: acctdisk, acctdusg,	acct(1M)
per-process accounting/	acctcms: command summary from	acctcms(1M)
process accounting file(s).	acctcom: search and print	acctcom(1)
connect-time accounting.	acctcon: acctcon1, acctcon2:	acctcon(1M)
connect-time/ acctcon:	acctcon1, acctcon2:	acctcon(1M)
acctcon: acctcon1,	acctcon2: connect-time/	acctcon(1M)
acctwtmp: overview of/ acct:	acctdisk, acctdusg, accton,	acct(1M)
overview of/ acct: acctdisk,	acctdusg, accton, acctwtmp:	acct(1M)
accounting files.	acctmerg: merge or add total	acctmerg(1M)
acct: acctdisk, acctdusg,	accton, acctwtmp: overview of/	acct(1M)
process accounting.	acctprc: acctprc1, acctprc2:	acctprc(1M)
accounting. acctprc:	acctprc1, acctprc2: process	acctprc(1M)
acctprc: acctprc1,	acctprc2: process accounting.	acctprc(1M)
dodisk, lastlogin, monacct,/	acctsh: chargefee, ckpacct,	acctsh(1M)
acctdisk, acctdusg, accton,	acctwtmp: overview of/ acct:	acct(1M)
killall: kill all	active processes.	killall(1M)
sag: system	activity graph.	sag(1G)
sar: sa1, sa2, sadc: system	activity report package.	sar(1M)
sar: system	activity reporter.	sar(1)
report process data and system	activity. /time a command;	timex(1)
device driver/ idinstall:	add, delete, update, or get	idinstall(1M)
acctmerg: merge or	add total accounting files.	acctmerg(1M)
Administration interface.	adm: invoke the AT&T	adm(1)
adm: invoke the AT&T	Administration interface.	adm(1)
network listener service	administration. nlsadmin:	nlsadmin(1M)
rfadmin: Remote File Sharing	administration.	rfadmin(1M)
uadmin:	administrative control.	uadmin(1M)
swap: swap	administrative interface.	swap(1M)
remote access.	adv: advertise a directory for	adv(1M)
remote access. adv:	advertise a directory for	adv(1M)

fumount: forced unmount of an	advertised resource.	fumount(1M)
accept, reject:	allow or prevent LP requests.	accept(1M)
fsba: file system block	analyzer.	fsba(1M)
sort: sort	and/or merge files.	sort(1)
introduction to commands and	application programs. intro:	intro(1)
language. bc:	arbitrary-precision arithmetic	bc(1)
tar: tape file	archiver.	tar(1)
cpio: copy file	archives in and out.	cpio(1)
command. xargs: construct	argument list(s) and execute	xargs(1)
expr: evaluate	arguments as an expression.	expr(1)
echo: echo	arguments.	echo(1)
bc: arbitrary-precision	arithmetic language.	bc(1)
expr: evaluate arguments	as an expression.	expr(1)
	asy: asynchronous serial port.	asy(7)
asy:	asynchronous serial port.	asy(7)
a later time.	at, batch: execute commands at	at(1)
adm: invoke the	AT&T Administration interface.	adm(1)
xt: multiplexed tty driver for	AT&T windowing terminals.	xt(7)
tunable parameter. idtune:	attempts to set value of a	idtune(1M)
login password and password	attributes. passwd: change	passwd(1M)
wait:	await completion of process.	wait(1)
processing language.	awk: pattern scanning and	awk(1)
backup: performs	backup functions.	backup(1M)
functions.	backup: performs backup	backup(1M)
	banner: make posters.	banner(1)
join: relational data	base operator.	join(1)
or query terminfo data	base. /initialize a terminal	tput(1)
(visual) display editor	based on ex. /screen-oriented	vi(1)
portions of path names.	basename, dirname: deliver	basename(1)
later time. at,	batch: execute commands at a	at(1)
arithmetic language.	bc: arbitrary-precision	bc(1)
initialization/ brc,	bcheckrc: system	brc(1M)
	bdiff: big diff.	bdiff(1)
	bfs: big file scanner.	bfs(1)
fsba: file system	block analyzer.	fsba(1M)
sum: print checksum and	block count of a file.	sum(1)
sync: update the super	block.	sync(1M)
df: report number of free disk	blocks and i-nodes.	df(1M)
memory image to/ sysdump:	boot option to dump system	sysdump(8)
initialization procedures.	brc, bcheckrc: system	brc(1M)
idbuild:	build new UNIX system kernel.	idbuild(1M)
mknod:	build special file.	mknod(1M)
disk: random access	bulk storage medium.	disk(7)
	cal: print calendar.	cal(1)
dc: desk	calculator.	dc(1)
cal: print	calendar.	cal(1)
	calendar: reminder service.	calendar(1)
cu:	call another UNIX system.	cu(1C)
to an LP line printer. lp,	cancel: send/cancel requests	lp(1)
description into a terminfo/	captoinfo: convert a termcap	captoinfo(1M)
streamer interface. qt: QIC	cartridge magnetic tape	qt(7)
text editor (variant of ex for	casual users). edit:	edit(1)
files.	cat: concatenate and print	cat(1)
	cd: change working directory.	cd(1)
conversion/ chrtbl: generate	character classification and	chrtbl(1M)
fgrep: search a file for a	character string.	fgrep(1)
tr: translate	characters.	tr(1)
lastlogin, monacct,/ acctsh:	chargefee, ckpacct, dodisk,	acctsh(1M)
fsck, dfsck:	check and repair file systems.	fsck(1M)

permissions file. uucheck:	check the uucp directories and	uucheck(1M)
grpck: password/group file	checkers. pwck,	pwck(1M)
file. sum: print	checksum and block count of a	sum(1)
chown,	chgrp: change owner or group.	chown(1)
	chmod: change mode.	chmod(1)
group.	chown, chgrp: change owner or	chown(1)
for a command.	chroot: change root directory	chroot(1M)
classification and conversion/	chrtbl: generate character	chrtbl(1M)
monacct,/ acctsh: chargefee,	ckpacct, dodisk, lastlogin,	acctsh(1M)
chrtbl: generate character	classification and conversion/	chrtbl(1M)
strclean: STREAMS error logger	cleanup program.	strclean(1M)
uucp spool directory	clean-up. uucleanup:	uucleanup(1M)
clri:	clear i-node.	clri(1M)
cron:	clock daemon.	cron(1M)
rtc: real time	clock interface.	rtc(7)
on a STREAMS driver.	clone: open any minor device	clone(7)
	clri: clear i-node.	clri(1M)
cram 7:	CMOS RAM interface.	cram(7)
	cmp: compare two files.	cmp(1)
line-feeds.	col: filter reverse	col(1)
common to two sorted files.	comm: select or reject lines	comm(1)
nice: run a	command at low priority.	nice(1)
change root directory for a	command. chroot:	chroot(1M)
env: set environment for	command execution.	env(1)
UNIX system to UNIX system	command execution. uux:	uux(1C)
quits. nohup: run a	command immune to hangups and	nohup(1)
getopt: parse	command options.	getopt(1)
getopts, getoptcvt: parse	command options.	getopts(1)
/shell, the standard/restricted	command programming language.	sh(1)
and system/ timex: time a	command; report process data	timex(1)
uuxqt: execute remote	command requests.	uuxqt(1M)
per-process/ acctcms:	command summary from	acctcms(1M)
test: condition evaluation	command.	test(1)
time: time a	command.	time(1)
argument list(s) and execute	command. xargs: construct	xargs(1)
and miscellaneous accounting	commands. /of accounting	acct(1M)
intro: introduction to	commands and application/	intro(1)
at, batch: execute	commands at a later time.	at(1)
multiuser/ rc2: run	commands performed for	rc2(1M)
operating system. rc0: run	commands performed to stop the	rc0(1M)
streamio: STREAMS ioctl	commands.	streamio(7)
comm: select or reject lines	common to two sorted files.	comm(1)
ipcs: report interprocess	communication facilities/	ipcs(1)
diff: differential file	comparator.	diff(1)
descriptions. infocmp:	compare or print out terminfo	infocmp(1M)
cmp:	compare two files.	cmp(1)
diff3: 3-way differential file	comparison.	diff3(1)
dircmp: directory	comparison.	dircmp(1)
driver. tp4: Intel ISO TC4	compatible TLI network device	tp4(7)
tic: terminfo	compiler.	tic(1M)
wait: await	completion of process.	wait(1)
pack, pcat, unpack:	compress and expand files.	pack(1)
cat:	concatenate and print files.	cat(1)
test:	condition evaluation command.	test(1)
update, or get device driver	configuration data. /delete,	idinstall(1M)
system. lpadmin:	configure the LP spooling	lpadmin(1M)
fwtmp, wtmpfix: manipulate	connect accounting records.	fwtmp(1M)
acctcon: acctcon1, acctcon2:	connect-time accounting.	acctcon(1M)
	console: console interface.	console(7)

display: system	console display.	display(7)
console:	console interface.	console(7)
keyboard: system	console keyboard.	keyboard(7)
mkfs:	construct a file system.	mkfs(1M)
mkfs:	construct a file system.	mkfs(1M)
execute command. xargs:	construct argument list(s) and	xargs(1)
nroff/troff, tbl, and eqn	constructs. deroff: remove	deroff(1)
debugging on. Uutry: try to	contact remote system with	Uutry(1M)
idmkinit: reads files	containing specifications.	idmkinit(1M)
ls: list	contents of directory.	ls(1)
csplit:	context split.	csplit(1)
device. tapecntl: tape	control for QIC-24/QIC-02 tape	tapecntl(1)
init, telinit: process	control initialization.	init(1M)
uadmin: administrative	control.	uadmin(1M)
uucp status inquiry and job	control. uustat:	uustat(1C)
interface. tty:	controlling terminal	tty(7)
units:	conversion program.	units(1)
character classification and	conversion tables. /generate	chrtbl(1M)
into a terminfo/ captoinfo:	convert a termcap description	captoinfo(1M)
dd:	convert and copy a file.	dd(1M)
timod: Transport Interface	cooperating STREAMS module.	timod(7)
dd: convert and	copy a file.	dd(1M)
cpio:	copy file archives in and out.	cpio(1)
access time. dcopy:	copy file systems for optimal	dcopy(1M)
cp, ln, mv:	copy, link, or move files.	cp(1)
volcopy: make literal	copy of file system.	volcopy(1M)
UNIX system to UNIX system	copy. uucp, uulog, uuname:	uucp(1C)
system to UNIX system file	copy. /uupick: public UNIX	uuto(1C)
mem, kmem:	core memory.	mem(7)
sum: print checksum and block	count of a file.	sum(1)
wc: word	count.	wc(1)
move files.	cp, ln, mv: copy, link, or	cp(1)
and out.	cpio: copy file archives in	cpio(1)
	cram 7: CMOS RAM interface.	cram(7)
	crash: examine system images.	crash(1M)
partition table. fdisk:	create or modify hard disk	fdisk(1M)
	cron: clock daemon.	cron(1M)
crontab: user	crontab file.	crontab(1)
	crontab: user crontab file.	crontab(1)
pg: file perusal filter for	CRTs.	pg(1)
	crypt: encode/decode.	crypt(1)
	csplit: context split.	csplit(1)
terminal.	ct: spawn getty to a remote	ct(1C)
	cu: call another UNIX system.	cu(1C)
rename login entry to show	current layer. relogin:	relogin(1M)
uname: print name of	current UNIX system.	uname(1)
spline: interpolate smooth	curve.	spline(1G)
of each line of a file.	cut: cut out selected fields	cut(1)
each line of a file. cut:	cut out selected fields of	cut(1)
cron: clock	daemon.	cron(1M)
rfudaemon: Remote File Sharing	daemon process.	rfudaemon(1M)
strerr: STREAMS error logger	daemon.	strerr(1M)
runacct: run	daily accounting.	runacct(1M)
/handle special functions of	DASI 300 and 300s terminals.	300(1)
special functions of the	DASI 450 terminal. /handle	450(1)
/time a command; report process	data and system activity.	timex(1)
join: relational	data base operator.	join(1)
a terminal or query terminfo	data base. tput: initialize	tput(1)
generate disk accounting	data by user ID. diskusg:	diskusg(1M)

device driver configuration	data. /delete, update, or get	idinstall(1M)
date: print and set the	date.	date(1)
	date: print and set the date.	date(1)
	dc: desk calculator.	dc(1)
optimal access time.	dcopy: copy file systems for	dcopy(1M)
	dd: convert and copy a file.	dd(1M)
fsdb: file system	debugger.	fsdb(1M)
to contact remote system with	debugging on. Uutry: try	Uutry(1M)
sysdef: output system	definition.	sysdef(1M)
driver/ idinstall: add,	delete, update, or get device	idinstall(1M)
names. basename, dirname:	deliver portions of path	basename(1)
file. tail:	deliver the last part of a	tail(1)
mesg: permit or	deny messages.	mesg(1)
tbl, and eqn constructs.	deroff: remove nroff/troff,	deroff(1)
description into a terminfo	description. /a termcap	captoinfo(1M)
captoinfo: convert a termcap	description into a terminfo/	captoinfo(1M)
compare or print out terminfo	descriptions. infocmp:	infocmp(1M)
dc:	desk calculator.	dc(1)
identifier. fstyp:	determine file system	fstyp(1M)
file:	determine file type.	file(1)
/add, delete, update, or get	device driver configuration/	idinstall(1M)
ISO TC4 compatible TLI network	device driver. tp4: Intel	tp4(7)
devnm:	device name.	devnm(1M)
clone: open any minor	device on a STREAMS driver.	clone(7)
control for QIC-24/QIC-02 tape	device. tapecntl: tape	tapecntl(1)
	devnm: device name.	devnm(1M)
blocks and i-nodes.	df: report number of free disk	df(1M)
systems. fsck,	dfsck: check and repair file	fsck(1M)
bdiff: big	diff.	bdiff(1)
comparator.	diff: differential file	diff(1)
comparison.	diff3: 3-way differential file	diff3(1)
sdiff: side-by-side	difference program.	sdiff(1)
diff:	differential file comparator.	diff(1)
diff3: 3-way	differential file comparison.	diff3(1)
	dircmp: directory comparison.	dircmp(1)
file. uucheck: check the uucp	directories and permissions	uucheck(1M)
link and unlink files and	directories. link, unlink:	link(1M)
mkdir: make	directories.	mkdir(1)
rm, rmdir: remove files or	directories.	rm(1)
cd: change working	directory.	cd(1)
uucleanup: uucp spool	directory clean-up.	uucleanup(1M)
dircmp:	directory comparison.	dircmp(1)
chroot: change root	directory for a command.	chroot(1M)
adv: advertise a	directory for remote access.	adv(1M)
ls: list contents of	directory.	ls(1)
mvdir: move a	directory.	mvdir(1M)
pwd: working	directory name.	pwd(1)
restore file to original	directory. restore:	restore(1M)
path names. basename,	dirname: deliver portions of	basename(1)
printers. enable,	disable: enable/disable LP	enable(1)
type, modes, speed, and line	discipline. /set terminal	getty(1M)
type, modes, speed, and line	discipline. /set terminal	uugetty(1M)
fusage:	disk access profiler.	fusage(1M)
ID. diskusg: generate	disk accounting data by user	diskusg(1M)
df: report number of free	disk blocks and i-nodes.	df(1M)
fd: diskette (floppy	disk).	fd(7)
hd: hard (fixed)	disk.	hd(7)
mkpart:	disk maintenance utility.	mkpart(1M)
fdisk: create or modify hard	disk partition table.	fdisk(1M)

diskadd:	disk partitioning utility.	diskadd(1M)
storage medium.	disk: random access bulk	disk(7)
format: format floppy	disk tracks.	format(1M)
du: summarize	disk usage.	du(1M)
utility.	diskadd: disk partitioning	diskadd(1M)
fd:	diskette (floppy disk).	fd(7)
system memory image to floppy	disk(s). /boot option to dump	sysdump(8)
accounting data by user ID.	diskusg: generate disk	diskusg(1M)
display: system console	display.	display(7)
/screen-oriented (visual)	display editor based on ex.	vi(1)
displaypkg:	display installed packages.	displaypkg(1)
information. rmntstat:	display mounted resource	rmntstat(1M)
display.	display: system console	display(7)
packages.	displaypkg: display installed	displaypkg(1)
object downloader for the 5620	DMD terminal. wtinit:	wtinit(1M)
Sharing domain and network/	dname: Print Remote File	dname(1M)
acctsh: chargefee, ckpacct,	dodisk, lastlogin, monacct,/	acctsh(1M)
whodo: who is	doing what.	whodo(1M)
/Print Remote File Sharing	domain and network names.	dname(1M)
terminal. wtinit: object	downloader for the 5620 DMD	wtinit(1M)
graph:	draw a graph.	graph(1G)
any minor device on a STREAMS	driver. clone: open	clone(7)
/delete, update, or get device	driver configuration data.	idinstall(1M)
xt: multiplexed tty	driver for AT&T windowing/	xt(7)
xtd: extract and print xt	driver link structure.	xtd(1M)
xtt: extract and print xt	driver packet traces.	xtt(1M)
xts: extract and print xt	driver statistics.	xts(1M)
sxt: pseudo-device	driver.	sxt(7)
compatible TLI network device	driver. tp4: Intel ISO TC4	tp4(7)
	du: summarize disk usage.	du(1M)
od: octal	dump.	od(1)
sysdump: boot option to	dump system memory image to/	sysdump(8)
echo:	echo arguments.	echo(1)
	echo: echo arguments.	echo(1)
	ed, red: text editor.	ed(1)
ex for casual users).	edit: text editor (variant of	edit(1)
/(visual) display	editor based on ex.	vi(1)
ed, red: text	editor.	ed(1)
ex: text	editor.	ex(1)
sed: stream	editor.	sed(1)
casual users). edit: text	editor (variant of ex for	edit(1)
pattern using full regular/	egrep: search a file for a	egrep(1)
enable/disable LP printers.	enable, disable:	enable(1)
enable, disable:	enable/disable LP printers.	enable(1)
crypt:	encode/decode.	crypt(1)
makekey: generate	encryption key.	makekey(1)
relogin: rename login	entry to show current layer.	relogin(1M)
command execution.	env: set environment for	env(1)
execution. env: set	environment for command	env(1)
performed for multiuser	environment. /run commands	rc2(1M)
remove nroff/troff, tbl, and	eqn constructs. deroff:	deroff(1)
strclean: STREAMS	error logger cleanup program.	strclean(1M)
strerr: STREAMS	error logger daemon.	strerr(1M)
log: interface to STREAMS	error logging and event/	log(7)
hashcheck: find spelling	errors. /hashmake, spellin,	spell(1)
setmnt:	establish mount table.	setmnt(1M)
expression. expr:	evaluate arguments as an	expr(1)
test: condition	evaluation command.	test(1)
to STREAMS error logging and	event tracing. log: interface	log(7)

edit: text editor (variant of	ex for casual users).	edit(1)
	ex: text editor.	ex(1)
display editor based on	ex. /screen-oriented (visual)	vi(1)
crash:	examine system images.	crash(1M)
construct argument list(s) and	execute command. xargs:	xargs(1)
time. at, batch:	execute commands at a later	at(1)
requests. uuxqt:	execute remote command	uuxqt(1M)
set environment for command	execution. env:	env(1)
sleep: suspend	execution for an interval.	sleep(1)
system to UNIX system command	execution. uux: UNIX	uux(1C)
pcat, unpack: compress and	expand files. pack,	pack(1)
expression.	expr: evaluate arguments as an	expr(1)
expr: evaluate arguments as an	expression.	expr(1)
a pattern using full regular	expressions. /a file for	egrep(1)
link structure. xtd:	extract and print xt driver	xtd(1M)
packet traces. xtt:	extract and print xt driver	xtt(1M)
statistics. xts:	extract and print xt driver	xts(1M)
factors of a number.	factor: obtain the prime	factor(1)
factor: obtain the prime	factors of a number.	factor(1)
true,	false: provide truth values.	true(1)
	fd: diskette (floppy disk).	fd(7)
disk partition table.	fdisk: create or modify hard	fdisk(1M)
statistics for a file system.	ff: list file names and	ff(1M)
character string.	fgrep: search a file for a	fgrep(1)
tar: tape	file archiver.	tar(1)
cpio: copy	file archives in and out.	cpio(1)
pwck, grpck: password/group	file checkers.	pwck(1M)
diff: differential	file comparator.	diff(1)
diff3: 3-way differential	file comparison.	diff3(1)
UNIX system to UNIX system	file copy. /uupick: public	uuto(1C)
crontab: user crontab	file.	crontab(1)
fields of each line of a	file. cut: cut out selected	cut(1)
dd: convert and copy a	file.	dd(1M)
	file: determine file type.	file(1)
fgrep: search a	file for a character string.	fgrep(1)
grep: search a	file for a pattern.	grep(1)
regular/ egrep: search a	file for a pattern using full	egrep(1)
split: split a	file into pieces.	split(1)
mknod: build special	file.	mknod(1M)
a file system. ff: list	file names and statistics for	ff(1M)
change the format of a text	file. newform:	newform(1)
null: the null	file.	null(7)
/identify processes using a	file or file structure.	fuser(1M)
or subsequent lines of one	file. /lines of several files	paste(1)
pg:	file perusal filter for CRTs.	pg(1)
bfs: big	file scanner.	bfs(1)
rfadmin: Remote	File Sharing administration.	rfadmin(1M)
rfudaemon: Remote	File Sharing daemon process.	rfudaemon(1M)
network/ dname: Print Remote	File Sharing domain and	dname(1M)
rfpasswd: change Remote	File Sharing host password.	rfpasswd(1M)
query. nsquery: Remote	File Sharing name server	nsquery(1M)
shell/ rfuadmin: Remote	File Sharing notification	rfuadmin(1M)
unadv: unadvertise a Remote	File Sharing resource.	unadv(1M)
/mount, unmount Remote	File Sharing resources.	rmountall(1M)
rfstart: start Remote	File Sharing.	rfstart(1M)
mapping. idload: Remote	File Sharing user and group	idload(1M)
processes using a file or	file structure. /identify	fuser(1M)
checksum and block count of a	file. sum: print	sum(1)
fsba:	file system block analyzer.	fsba(1M)

fsdb:	file system debugger.	fsdb(1M)
names and statistics for a	file system. ff: list file	ff(1M)
fstyp: determine	file system identifier.	fstyp(1M)
mkfs: construct a	file system.	mkfs(1M)
mkfs: construct a	file system.	mkfs(1M)
fsstat: report	file system status.	fsstat(1M)
volcopy: make literal copy of	file system.	volcopy(1M)
/umount: mount and unmount	file systems and remote/	mount(1M)
access time. dcopy: copy	file systems for optimal	dcopy(1M)
fsck, dfsck: check and repair	file systems.	fsck(1M)
labelit: provide labels for	file systems.	labelit(1M)
mount, unmount multiple	file systems. /umountall:	mountall(1M)
deliver the last part of a	file. tail:	tail(1)
restore: restore	file to original directory.	restore(1M)
and modification times of a	file. touch: update access	touch(1)
uucp system. uucico:	file transport program for the	uucico(1M)
/the scheduler for the uucp	file transport program.	uusched(1M)
file: determine	file type.	file(1)
report repeated lines in a	file. uniq:	uniq(1)
directories and permissions	file. uucheck: check the uucp	uucheck(1M)
umask: set	file-creation mode mask.	umask(1)
and print process accounting	file(s). acctcom: search	acctcom(1)
merge or add total accounting	files. acctmerg:	acctmerg(1M)
link, unlink: link and unlink	files and directories.	link(1M)
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)
idmkinit: reads	files containing/	idmkinit(1M)
ln, mv: copy, link, or move	files. cp,	cp(1)
find: find	files.	find(1)
intro: introduction to special	files.	intro(7)
passmgmt: password	files management.	passmgmt(1M)
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)
sort: sort and/or merge	files.	sort(1)
pg: file perusal	filter for CRTs.	pg(1)
greek: select terminal	filter.	greek(1)
nl: line-numbering	filter.	nl(1)
col:	filter reverse line-feeds.	col(1)
tplot: graphics	filters.	tplot(1G)
find:	find files.	find(1)
	find: find files.	find(1)
hashmake, spellin, hashcheck:	find spelling errors. spell,	spell(1)
tee: pipe	fitting.	tee(1)
hd: hard	(fixed) disk.	hd(7)
fd: diskette	(floppy disk).	fd(7)
format: format	floppy disk tracks.	format(1M)
to dump system memory image to	floppy disk(s). /boot option	sysdump(8)
advertised resource. fumount:	forced unmount of an	fumount(1M)
format:	format floppy disk tracks.	format(1M)
tracks.	format: format floppy disk	format(1M)
newform: change the	format of a text file.	newform(1)
df: report number of	free disk blocks and i-nodes.	df(1M)
idspace: investigates	free space.	idspace(1M)
ncheck: generate path names	from i-numbers.	ncheck(1M)
acctcms: command summary	from per-process accounting/	acctcms(1M)
analyzer.	fsba: file system block	fsba(1M)

file systems.	fsck, dfsck: check and repair	fsck(1M)
	fsdb: file system debugger.	fsdb(1M)
status.	fsstat: report file system	fsstat(1M)
identifier.	fstyp: determine file system	fstyp(1M)
/a file for a pattern using	full regular expressions.	egrep(1)
advertised resource.	fumount: forced unmount of an	fumount(1M)
backup: performs backup	functions.	backup(1M)
300, 300s: handle special	functions of DASI 300 and 300s/	300(1)
terminals. hp: handle special	functions of Hewlett-Packard	hp(1)
terminal. 450: handle special	functions of the DASI 450	450(1)
	fusage: disk access profiler.	fusage(1M)
using a file or file/	fuser: identify processes	fuser(1M)
connect accounting records.	fwtmp, wtmpfix: manipulate	fwtmp(1M)
classification and/ chrtbl:	generate character	chrtbl(1M)
by user ID. diskusg:	generate disk accounting data	diskusg(1M)
makekey:	generate encryption key.	makekey(1)
i-numbers. ncheck:	generate path names from	ncheck(1M)
add, delete, update, or	get device driver/ idinstall:	idinstall(1M)
logname:	get login name.	logname(1)
value. machid: i386:	get processor type truth	machid(1)
tty:	get the name of the terminal.	tty(1)
	getopt: parse command options.	getopt(1)
options. getopts,	getoptcvt: parse command	getopts(1)
command options.	getopts, getoptcvt: parse	getopts(1)
modes, speed, and line/	getty: set terminal type,	getty(1M)
ct: spawn	getty to a remote terminal.	ct(1C)
	graph: draw a graph.	graph(1G)
graph: draw a	graph.	graph(1G)
sag: system activity	graph.	sag(1G)
tplot:	graphics filters.	tplot(1G)
	greek: select terminal filter.	greek(1)
pattern.	grep: search a file for a	grep(1)
chown, chgrp: change owner or	group.	chown(1)
id: print user and	group IDs and names.	id(1M)
Remote File Sharing user and	group mapping. idload:	idload(1M)
newgrp: log in to a new	group.	newgrp(1M)
checkers. pwck,	grpck: password/group file	pwck(1M)
DASI 300 and 300s/ 300, 300s:	handle special functions of	300(1)
Hewlett-Packard/ hp:	handle special functions of	hp(1)
the DASI 450 terminal. 450:	handle special functions of	450(1)
nohup: run a command immune to	hangups and quits.	nohup(1)
fdisk: create or modify	hard disk partition table.	fdisk(1M)
hd:	hard (fixed) disk.	hd(7)
spell, hashmake, spellin,	hashcheck: find spelling/	spell(1)
find spelling errors. spell,	hashmake, spellin, hashcheck:	spell(1)
	hd: hard (fixed) disk.	hd(7)
/handle special functions of	Hewlett-Packard terminals.	hp(1)
change Remote File Sharing	host password. rfpasswd:	rfpasswd(1M)
of Hewlett-Packard terminals.	hp: handle special functions	hp(1)
value. machid:	i386: get processor type truth	machid(1)
disk accounting data by user	ID. diskusg: generate	diskusg(1M)
set, or shared memory	id. /message queue, semaphore	ipcrm(1)
and names.	id: print user and group IDs	id(1M)
kernel.	idbuild: build new UNIX system	idbuild(1M)
information.	idcheck: returns selected	idcheck(1M)
fstyp: determine file system	identifier.	fstyp(1M)
file or file/ fuser:	identify processes using a	fuser(1M)
update, or get device driver/	idinstall: add, delete,	idinstall(1M)
user and group mapping.	idload: Remote File Sharing	idload(1M)

containing specifications.	idmkinit: reads files	idmkinit(1M)
reads specifications of/	idmknod: removes nodes and	idmknod(1M)
id: print user and group	IDs and names.	id(1M)
space.	idspace: investigates free	idspace(1M)
of a tunable parameter.	idtune: attempts to set value	idtune(1M)
option to dump system memory	image to floppy disk(s). /boot	sysdump(8)
crash: examine system	images.	crash(1M)
nohup: run a command	immune to hangups and quits.	nohup(1)
terminfo descriptions.	infocmp: compare or print out	infocmp(1M)
initialization.	init, telinit: process control	init(1M)
init, telinit: process control	initialization.	init(1M)
brc, bcheckrc: system	initialization procedures.	brc(1M)
terminfo data base. tput:	initialize a terminal or query	tput(1)
clri: clear	i-node.	clri(1M)
number of free disk blocks and	i-nodes. df: report	df(1M)
uustat: uucp status	inquiry and job control.	uustat(1C)
pwconv:	install and update.	pwconv(1M)
installpkg:	install package.	installpkg(1)
removepkg: remove	installed package.	removepkg(1)
displaypkg: display	installed packages.	displaypkg(1)
	installpkg: install package.	installpkg(1)
network device driver. tp4:	Intel ISO TC4 compatible TLI	tp4(7)
system. mailx:	interactive message processing	mailx(1)
invoke the AT&T Administration	interface. adm:	adm(1)
console: console	interface.	console(7)
module. timod: Transport	Interface cooperating STREAMS	timod(7)
cram 7: CMOS RAM	interface.	cram(7)
lp: parallel printer	interface.	lp(7)
magnetic tape streamer	interface. qt: QIC cartridge	qt(7)
STREAMS/ tirdwr: Transport	Interface read/write interface	tirdwr(7)
rtc: real time clock	interface.	rtc(7)
/Transport Interface read/write	interface STREAMS module.	tirdwr(7)
swap: swap administrative	interface.	swap(1M)
termio: general terminal	interface.	termio(7)
logging and event/ log:	interface to STREAMS error	log(7)
tty: controlling terminal	interface.	tty(7)
spline:	interpolate smooth curve.	spline(1G)
facilities/ ipcs: report	interprocess communication	ipcs(1)
suspend execution for an	interval. sleep:	sleep(1)
commands and application/	intro: introduction to	intro(1)
files.	intro: introduction to special	intro(7)
maintenance procedures.	intro: introduction to system	intro(8)
application programs. intro:	introduction to commands and	intro(1)
intro:	introduction to special files.	intro(7)
maintenance/ intro:	introduction to system	intro(8)
generate path names from	i-numbers. ncheck:	ncheck(1M)
idspace:	investigates free space.	idspace(1M)
interface. adm:	invoke the AT&T Administration	adm(1)
streamio: STREAMS	ioctl commands.	streamio(7)
semaphore set, or shared/	ipcrm: remove a message queue,	ipcrm(1)
communication facilities/	ipcs: report interprocess	ipcs(1)
terminal state.	ismpx: return windowing	ismpx(1)
device driver. tp4: Intel	ISO TC4 compatible TLI network	tp4(7)
news: print news	items.	news(1)
operator.	join: relational data base	join(1)
windowing terminal.	jterm: reset layer of	jterm(1)
	jwin: print size of layer.	jwin(1)
idbuild: build new UNIX system	kernel.	idbuild(1M)
makekey: generate encryption	key.	makekey(1)

keyboard: system console	keyboard.	keyboard(7)
keyboard.	keyboard: system console	keyboard(7)
killall:	kill all active processes.	killall(1M)
	kill: terminate a process.	kill(1)
processes.	killall: kill all active	killall(1M)
mem,	kmem: core memory.	mem(7)
file systems.	labelit: provide labels for	labelit(1M)
labelit: provide	labels for file systems.	labelit(1M)
scanning and processing	language. awk: pattern	awk(1)
arbitrary-precision arithmetic	language. bc:	bc(1)
scanning and processing	language. nawk: pattern	nawk(1)
command programming	language. /standard/restricted	sh(1)
/chargefee, ckpacct, dodisk,	lastlogin, monacct, nulladm,/	acctsh(1M)
jwin: print size of	layer.	jwin(1)
shl: shell	layer manager.	shl(1)
windowing terminals. layers:	layer multiplexer for	layers(1)
jterm: reset	layer of windowing terminal.	jterm(1)
login entry to show current	layer. relogin: rename	relogin(1M)
windowing terminals.	layers: layer multiplexer for	layers(1)
type, modes, speed, and	line discipline. /set terminal	getty(1M)
type, modes, speed, and	line discipline. /set terminal	uugetty(1M)
line: read one	line.	line(1)
out selected fields of each	line of a file. cut: cut	cut(1)
send/cancel requests to an LP	line printer. lp, cancel:	lp(1)
	line: read one line.	line(1)
col: filter reverse	line-feeds.	col(1)
nl:	line-numbering filter.	nl(1)
files. comm: select or reject	lines common to two sorted	comm(1)
uniq: report repeated	lines in a file.	uniq(1)
of several files or subsequent	lines of one file. /same lines	paste(1)
subsequent/ paste: merge same	lines of several files or	paste(1)
directories. link, unlink:	link and unlink files and	link(1M)
cp, ln, mv: copy,	link, or move files.	cp(1)
extract and print xt driver	link structure. xtd:	xtd(1M)
files and directories.	link, unlink: link and unlink	link(1M)
ls:	list contents of directory.	ls(1)
for a file system. ff:	list file names and statistics	ff(1M)
nlsadmin: network	listener service/	nlsadmin(1M)
xargs: construct argument	list(s) and execute command.	xargs(1)
volcopy: make	literal copy of file system.	volcopy(1M)
files. cp,	ln, mv: copy, link, or move	cp(1)
newgrp:	log in to a new group.	newgrp(1M)
error logging and event/	log: interface to STREAMS	log(7)
strclean: STREAMS error	logger cleanup program.	strclean(1M)
strerr: STREAMS error	logger daemon.	strerr(1M)
/interface to STREAMS error	logging and event tracing.	log(7)
layer. relogin: rename	login entry to show current	relogin(1M)
logname: get	login name.	logname(1)
attributes. passwd: change	login password and password	passwd(1M)
passwd: change	login password.	passwd(1)
	login: sign on.	login(1)
	logname: get login name.	logname(1)
nice: run a command at	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)
interface.	lp: parallel printer	lp(7)
disable: enable/disable	LP printers. enable,	enable(1)
reject: allow or prevent	LP requests. accept,	accept(1M)
/lpshut, lpmove: start/stop the	LP scheduler and move/	lpsched(1M)

lpadmin: configure the	LP spooling system.	lpadmin(1M)
lpstat: print	LP status information.	lpstat(1)
spooling system.	lpadmin: configure the LP	lpadmin(1M)
scheduler/ lpsched, lpshut,	lpmove: start/stop the LP	lpsched(1M)
start/stop the LP scheduler/	lpsched, lpshut, lpmove:	lpsched(1M)
LP scheduler and/ lpsched,	lpshut, lpmove: start/stop the	lpsched(1M)
information.	lpstat: print LP status	lpstat(1)
directory.	ls: list contents of	ls(1)
type truth value.	machid: i386: get processor	machid(1)
interface. qt: QIC cartridge	magnetic tape streamer	qt(7)
send mail to users or read	mail. mail, rmail:	mail(1)
users or read mail.	mail, rmail: send mail to	mail(1)
mail, rmail: send	mail to users or read mail.	mail(1)
processing system.	mailx: interactive message	mailx(1)
intro: introduction to system	maintenance procedures.	intro(8)
mkpart: disk	maintenance utility.	mkpart(1M)
mkdir:	make directories.	mkdir(1)
system. volcopy:	make literal copy of file	volcopy(1M)
banner:	make posters.	banner(1)
key.	makekey: generate encryption	makekey(1)
passmgmt: password files	management.	passmgmt(1M)
shl: shell layer	manager.	shl(1)
records. fwtmp, wtmpfix:	manipulate connect accounting	fwtmp(1M)
File Sharing user and group	mapping. idload: Remote	idload(1M)
umask: set file-creation mode	mask.	umask(1)
random access bulk storage	medium. disk:	disk(7)
	mem, kmem: core memory.	mem(7)
semaphore set, or shared	memory id. /a message queue,	ipcrm(1)
/boot option to dump system	memory image to floppy/	sysdump(8)
mem, kmem: core	memory.	mem(7)
sort: sort and/or	merge files.	sort(1)
files. acctmerg:	merge or add total accounting	acctmerg(1M)
files or subsequent/ paste:	merge same lines of several	paste(1)
	mesg: permit or deny messages.	mesg(1)
mailx: interactive	message processing system.	mailx(1)
or shared/ ipcrm: remove a	message queue, semaphore set,	ipcrm(1)
mesg: permit or deny	messages.	mesg(1)
strace: print STREAMS trace	messages.	strace(1M)
driver. clone: open any	minor device on a STREAMS	clone(7)
	mkdir: make directories.	mkdir(1)
	mkfs: construct a file system.	mkfs(1M)
	mkfs: construct a file system.	mkfs(1M)
	mknod: build special file.	mknod(1M)
utility.	mkpart: disk maintenance	mkpart(1M)
chmod: change	mode.	chmod(1)
umask: set file-creation	mode mask.	umask(1)
getty: set terminal type,	modes, speed, and line/	getty(1M)
uugetty: set terminal type,	modes, speed, and line/	uugetty(1M)
touch: update access and	modification times of a file.	touch(1)
table. fdisk: create or	modify hard disk partition	fdisk(1M)
Interface cooperating STREAMS	module. timod: Transport	timod(7)
read/write interface STREAMS	module. /Transport Interface	tirdwr(7)
/ckpacct, dodisk, lastlogin,	monacct, nulladm, prctmp,/	acctsh(1M)
and remote/ mount, umount:	mount and unmount file systems	mount(1M)
setmnt: establish	mount table.	setmnt(1M)
unmount file systems and/	mount, umount: mount and	mount(1M)
systems. mountall, umountall:	mount, unmount multiple file	mountall(1M)
rmountall, rumountall:	mount, unmount Remote File/	rmountall(1M)
unmount multiple file/	mountall, umountall: mount,	mountall(1M)

rmntstat: display	mounted resource information.	rmntstat(1M)
rmount: retry remote resource	mounts.	rmount(1M)
mvdir:	move a directory.	mvdir(1M)
cp, ln, mv: copy, link, or	move files.	cp(1)
the LP scheduler and	move requests. /start/stop	lpsched(1M)
/umountall: mount, unmount	multiple file systems.	mountall(1M)
AT&T windowing terminals. xt:	multiplexed tty driver for	xt(7)
terminals. layers: layer	multiplexer for windowing	layers(1)
run commands performed for	multiuser environment. rc2:	rc2(1M)
cp, ln,	mv: copy, link, or move files.	cp(1)
	mvdir: move a directory.	mvdir(1M)
processing language.	nawk: pattern scanning and	nawk(1)
from i-numbers.	ncheck: generate path names	ncheck(1M)
Intel ISO TC4 compatible TLI	network device driver. tp4:	tp4(7)
administration. nlsadmin:	network listener service	nlsadmin(1M)
Remote File Sharing domain and	network names. dname: Print	dname(1M)
a text file.	newform: change the format of	newform(1)
	newgrp: log in to a new group.	newgrp(1M)
news: print	news items.	news(1)
	news: print news items.	news(1)
priority.	nice: run a command at low	nice(1)
	nl: line-numbering filter.	nl(1)
service administration.	nlsadmin: network listener	nlsadmin(1M)
of nodes. idmknod: removes	nodes and reads specifications	idmknod(1M)
and reads specifications of	nodes. idmknod: removes nodes	idmknod(1M)
hangups and quits.	nohup: run a command immune to	nohup(1)
rfuadmin: Remote File Sharing	notification shell script.	rfuadmin(1M)
constructs. deroff: remove	nroff/troff, tbl, and eqn	deroff(1)
name server query.	nsquery: Remote File Sharing	nsquery(1M)
null: the	null file.	null(7)
	null: the null file.	null(7)
/dodisk, lastlogin, monacct,	nulladm, prctmp, prdaily,/	acctsh(1M)
DMD terminal. wtinit:	object downloader for the 5620	wtinit(1M)
number. factor:	obtain the prime factors of a	factor(1)
od:	octal dump.	od(1)
	od: octal dump.	od(1)
STREAMS driver. clone:	open any minor device on a	clone(7)
prf:	operating system profiler.	prf(7)
commands performed to stop the	operating system. rc0: run	rc0(1M)
join: relational data base	operator.	join(1)
dcopy: copy file systems for	optimal access time.	dcopy(1M)
image to floppy/ sysdump: boot	option to dump system memory	sysdump(8)
stty: set the	options for a terminal.	stty(1)
getopt: parse command	options.	getopt(1)
getoptcvt: parse command	options. getopts,	getopts(1)
restore: restore file to	original directory.	restore(1M)
sysdef:	output system definition.	sysdef(1M)
/acctdusg, accton, acctwtmp:	overview of accounting and/	acct(1M)
chown, chgrp: change	owner or group.	chown(1)
and expand files.	pack, pcat, unpack: compress	pack(1)
installpkg: install	package.	installpkg(1)
removepkg: remove installed	package.	removepkg(1)
sadc: system activity report	package. sar: sa1, sa2,	sar(1M)
displaypkg: display installed	packages.	displaypkg(1)
extract and print xt driver	packet traces. xtt:	xtt(1M)
4014 terminal. 4014:	paginator for the TEKTRONIX	4014(1)
lp:	parallel printer interface.	lp(7)
to set value of a tunable	parameter. idtune: attempts	idtune(1M)
getopt:	parse command options.	getopt(1)

getopts, getoptcvt:	parse command options.	getopts(1)
create or modify hard disk	partition table. fdisk:	fdisk(1M)
diskadd: disk	partitioning utility.	diskadd(1M)
management.	passmgmt: password files	passmgmt(1M)
and password attributes.	passwd: change login password	passwd(1M)
	passwd: change login password.	passwd(1)
passwd: change login	password and password/	passwd(1M)
change login password and	password attributes. passwd:	passwd(1M)
passmgmt:	password files management.	passmgmt(1M)
passwd: change login	password.	passwd(1)
Remote File Sharing host	password. rfpasswd: change	rfpasswd(1M)
pwck, grpck:	password/group file checkers.	pwck(1M)
several files or subsequent/	paste: merge same lines of	paste(1)
dirname: deliver portions of	path names. basename,	basename(1)
ncheck: generate	path names from i-numbers.	ncheck(1M)
grep: search a file for a	pattern.	grep(1)
processing language. awk:	pattern scanning and	awk(1)
processing language. nawk:	pattern scanning and	nawk(1)
egrep: search a file for a	pattern using full regular/	egrep(1)
expand files. pack,	pcat, unpack: compress and	pack(1)
rc2: run commands	performed for multiuser/	rc2(1M)
operating/ rc0: run commands	performed to stop the	rc0(1M)
backup:	performs backup functions.	backup(1M)
check the uucp directories and	permissions file. uucheck:	uucheck(1M)
mesg:	permit or deny messages.	mesg(1)
acctcms: command summary from	per-process accounting/	acctcms(1M)
pg: file	perusal filter for CRTs.	pg(1)
CRTs.	pg: file perusal filter for	pg(1)
split: split a file into	pieces.	split(1)
tee:	pipe fitting.	tee(1)
asy: asynchronous serial	port.	asy(7)
basename, dirname: deliver	portions of path names.	basename(1)
banner: make	posters.	banner(1)
	pr: print files.	pr(1)
/lastlogin, monacct, nulladm,	prctmp, prdaily, prtacct,/	acctsh(1M)
/monacct, nulladm, prctmp,	prdaily, prtacct, runacct,/	acctsh(1M)
accept, reject: allow or	prevent LP requests.	accept(1M)
profiler.	prf: operating system	prf(7)
profiler: prfld, prfstat,	prfdc, prfsnap, prfpr: UNIX/	profiler(1M)
prfsnap, prfpr:/ profiler:	prfld, prfstat, prfdc,	profiler(1M)
/prfstat, prfdc, prfsnap,	prfpr: UNIX system profiler.	profiler(1M)
/prfld, prfstat, prfdc,	prfsnap, prfpr: UNIX system/	profiler(1M)
prfpr: UNIX/ profiler: prfld,	prfstat, prfdc, prfsnap,	profiler(1M)
factor: obtain the	prime factors of a number.	factor(1)
date:	print and set the date.	date(1)
cal:	print calendar.	cal(1)
of a file. sum:	print checksum and block count	sum(1)
cat: concatenate and	print files.	cat(1)
pr:	print files.	pr(1)
lpstat:	print LP status information.	lpstat(1)
system. uname:	print name of current UNIX	uname(1)
news:	print news items.	news(1)
infocmp: compare or	print out terminfo/	infocmp(1M)
file(s). acctcom: search and	print process accounting	acctcom(1)
domain and network/ dname:	Print Remote File Sharing	dname(1M)
jwin:	print size of layer.	jwin(1)
strace:	print STREAMS trace messages.	strace(1M)
names. id:	print user and group IDs and	id(1M)
structure. xtd: extract and	print xt driver link	xtd(1M)

xtt: extract and	print xt driver packet traces.	xtt(1M)
xts: extract and	print xt driver statistics.	xts(1M)
lp: parallel	printer interface.	lp(7)
requests to an LP line	printer. /cancel: send/cancel	lp(1)
disable: enable/disable LP	printers. enable,	enable(1)
nice: run a command at low	priority.	nice(1)
acctprc: acctprc1, acctprc2:	process accounting.	acctprc(1M)
acctcom: search and print	process accounting file(s).	acctcom(1)
init, telinit:	process control/	init(1M)
timex: time a command; report	process data and system/	timex(1)
kill: terminate a	process.	kill(1)
Remote File Sharing daemon	process. rfudaemon:	rfudaemon(1M)
ps: report	process status.	ps(1)
wait: await completion of	process.	wait(1)
killall: kill all active	processes.	killall(1M)
structure. fuser: identify	processes using a file or file	fuser(1M)
awk: pattern scanning and	processing language.	awk(1)
nawk: pattern scanning and	processing language.	nawk(1)
mailx: interactive message	processing system.	mailx(1)
machid: i386: get	processor type truth value.	machid(1)
fusage: disk access	profiler.	fusage(1M)
prf: operating system	profiler.	prf(7)
prfdc, prfsnap, prfpr: UNIX/	profiler: prfld, prfstat,	profiler(1M)
prfsnap, prfpr: UNIX system	profiler. /prfstat, prfdc,	profiler(1M)
standard/restricted command	programming language. /the	sh(1)
systems. labelit:	provide labels for file	labelit(1M)
true, false:	provide truth values.	true(1)
/nulladm, prctmp, prdaily,	prtacct, runacct, shutacct,/	acctsh(1M)
	ps: report process status.	ps(1)
sxt:	pseudo-device driver.	sxt(7)
file checkers.	pwck, grpck: password/group	pwck(1M)
	pwconv: install and update.	pwconv(1M)
	pwd: working directory name.	pwd(1)
streamer interface. qt:	QIC cartridge magnetic tape	qt(7)
tapecntl: tape control for	QIC-24/QIC-02 tape device.	tapecntl(1)
tape streamer interface.	qt: QIC cartridge magnetic	qt(7)
File Sharing name server	query. nsquery: Remote	nsquery(1M)
tput: initialize a terminal or	query terminfo data base.	tput(1)
ipcrm: remove a message	queue, semaphore set, or/	ipcrm(1)
command immune to hangups and	quits. nohup: run a	nohup(1)
cram 7: CMOS	RAM interface.	cram(7)
medium. disk:	random access bulk storage	disk(7)
stop the operating system.	rc0: run commands performed to	rc0(1M)
for multiuser environment.	rc2: run commands performed	rc2(1M)
rmail: send mail to users or	read mail. mail,	mail(1)
line:	read one line.	line(1)
specifications. idmkinit:	reads files containing	idmkinit(1M)
idmknod: removes nodes and	reads specifications of nodes.	idmknod(1M)
tirdwr: Transport Interface	read/write interface STREAMS/	tirdwr(7)
from per-process accounting	records. /command summary	acctcms(1M)
manipulate connect accounting	records. fwtmp, wtmpfix:	fwtmp(1M)
ed,	red: text editor.	ed(1)
file for a pattern using full	regular expressions. /search a	egrep(1)
requests. accept,	reject: allow or prevent LP	accept(1M)
sorted files. comm: select or	reject lines common to two	comm(1)
join:	relational data base operator.	join(1)
show current layer.	relogin: rename login entry to	relogin(1M)
calendar:	reminder service.	calendar(1)
adv: advertise a directory for	remote access.	adv(1M)

uuxqt: execute	remote command requests.	uuxqt(1M)
administration. rfadmin:	Remote File Sharing	rfadmin(1M)
process. rfudaemon:	Remote File Sharing daemon	rfudaemon(1M)
network names. dname: Print	Remote File Sharing domain and	dname(1M)
password. rfpasswd: change	Remote File Sharing host	rfpasswd(1M)
server query. nsquery:	Remote File Sharing name	nsquery(1M)
notification shell/ rfuadmin:	Remote File Sharing	rfuadmin(1M)
unadv: unadvertise a	Remote File Sharing resource.	unadv(1M)
/rumountall: mount, unmount	Remote File Sharing resources.	rmountall(1M)
rfstart: start	Remote File Sharing.	rfstart(1M)
group mapping. idload:	Remote File Sharing user and	idload(1M)
rmount: retry	remote resource mounts.	rmount(1M)
and unmount file systems and	remote resources. /mount	mount(1M)
on. Uutry: try to contact	remote system with debugging	Uutry(1M)
ct: spawn getty to a	remote terminal.	ct(1C)
semaphore set, or/ ipcrm:	remove a message queue,	ipcrm(1)
rm, rmdir:	remove files or directories.	rm(1)
removepkg:	remove installed package.	removepkg(1)
eqn constructs. deroff:	remove nroff/troff, tbl, and	deroff(1)
package.	removepkg: remove installed	removepkg(1)
specifications of/ idmknod:	removes nodes and reads	idmknod(1M)
current layer. relogin:	rename login entry to show	relogin(1M)
fsck, dfsck: check and	repair file systems.	fsck(1M)
uniq: report	repeated lines in a file.	uniq(1)
fsstat:	report file system status.	fsstat(1M)
communication/ ipcs:	report interprocess	ipcs(1)
blocks and i-nodes. df:	report number of free disk	df(1M)
sa2, sadc: system activity	report package. sar: sa1,	sar(1M)
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)
sar: system activity	reporter.	sar(1)
reject: allow or prevent LP	requests. accept,	accept(1M)
the LP scheduler and move	requests. /lpmove: start/stop	lpsched(1M)
lp, cancel: send/cancel	requests to an LP line/	lp(1)
uuxqt: execute remote command	requests.	uuxqt(1M)
terminal. jterm:	reset layer of windowing	jterm(1)
unmount of an advertised	resource. fumount: forced	fumount(1M)
rmntstat: display mounted	resource information.	rmntstat(1M)
rmount: retry remote	resource mounts.	rmount(1M)
a Remote File Sharing	resource. unadv: unadvertise	unadv(1M)
file systems and remote	resources. /mount and unmount	mount(1M)
unmount Remote File Sharing	resources. /rumountall: mount,	rmountall(1M)
directory. restore:	restore file to original	restore(1M)
original directory.	restore: restore file to	restore(1M)
rmount:	retry remote resource mounts.	rmount(1M)
state. ismpx:	return windowing terminal	ismpx(1)
idcheck:	returns selected information.	idcheck(1M)
col: filter	reverse line-feeds.	col(1)
administration.	rfadmin: Remote File Sharing	rfadmin(1M)
Sharing host password.	rfpasswd: change Remote File	rfpasswd(1M)
Sharing.	rfstart: start Remote File	rfstart(1M)
notification shell script.	rfuadmin: Remote File Sharing	rfuadmin(1M)
daemon process.	rfudaemon: Remote File Sharing	rfudaemon(1M)
directories.	rm, rmdir: remove files or	rm(1)
read mail. mail,	rmail: send mail to users or	mail(1)
directories. rm,	rmdir: remove files or	rm(1)
resource information.	rmntstat: display mounted	rmntstat(1M)
mounts.	rmount: retry remote resource	rmount(1M)

unmount Remote File Sharing/	rmountall, rumountall: mount,	rmountall(1M)
chroot: change	root directory for a command.	chroot(1M)
standard/restricted/ sh,	rsh: shell, the	sh(1)
interface.	rtc: real time clock	rtc(7)
Remote File/ rmountall,	rumountall: mount, unmount	rmountall(1M)
nice:	run a command at low priority.	nice(1)
hangups and quits. nohup:	run a command immune to	nohup(1)
multiuser environment. rc2:	run commands performed for	rc2(1M)
the operating system. rc0:	run commands performed to stop	rc0(1M)
runacct:	run daily accounting.	runacct(1M)
	runacct: run daily accounting.	runacct(1M)
/prctmp, prdaily, prtacct,	runacct, shutacct, startup,/	acctsh(1M)
activity report package. sar:	sa1, sa2, sadc: system	sar(1M)
report package. sar: sa1,	sa2, sadc: system activity	sar(1M)
package. sar: sa1, sa2,	sadc: system activity report	sar(1M)
	sag: system activity graph.	sag(1G)
activity report package.	sar: sa1, sa2, sadc: system	sar(1M)
	sar: system activity reporter.	sar(1)
bfs: big file	scanner.	bfs(1)
language. awk: pattern	scanning and processing	awk(1)
language. nawk: pattern	scanning and processing	nawk(1)
/lpmove: start/stop the LP	scheduler and move requests.	lpsched(1M)
transport/ uusched: the	scheduler for the uucp file	uusched(1M)
display/ vi, view, vedit:	screen-oriented (visual)	vi(1)
Sharing notification shell	script. rfuadmin: Remote File	rfuadmin(1M)
program.	sdiff: side-by-side difference	sdiff(1)
string. fgrep:	search a file for a character	fgrep(1)
grep:	search a file for a pattern.	grep(1)
using full regular/ egrep:	search a file for a pattern	egrep(1)
accounting file(s). acctcom:	search and print process	acctcom(1)
	sed: stream editor.	sed(1)
to two sorted files. comm:	select or reject lines common	comm(1)
greek:	select terminal filter.	greek(1)
of a file. cut: cut out	selected fields of each line	cut(1)
idcheck: returns	selected information.	idcheck(1M)
ipcrm: remove a message queue,	semaphore set, or shared/	ipcrm(1)
mail. mail, rmail:	send mail to users or read	mail(1)
line printer. lp, cancel:	send/cancel requests to an LP	lp(1)
asy: asynchronous	serial port.	asy(7)
Remote File Sharing name	server query. nsquery:	nsquery(1M)
/a message queue, semaphore	set, or shared memory id.	ipcrm(1)
	setmnt: establish mount table.	setmnt(1M)
standard/restricted command/	sh, rsh: shell, the	sh(1)
queue, semaphore set, or	shared memory id. /a message	ipcrm(1)
rfadmin: Remote File	Sharing administration.	rfadmin(1M)
rfudaemon: Remote File	Sharing daemon process.	rfudaemon(1M)
dname: Print Remote File	Sharing domain and network/	dname(1M)
rfpasswd: change Remote File	Sharing host password.	rfpasswd(1M)
nsquery: Remote File	Sharing name server query.	nsquery(1M)
script. rfuadmin: Remote File	Sharing notification shell	rfuadmin(1M)
unadvertise a Remote File	Sharing resource. unadv:	unadv(1M)
/mount, unmount Remote File	Sharing resources.	rmountall(1M)
rfstart: start Remote File	Sharing.	rfstart(1M)
mapping. idload: Remote File	Sharing user and group	idload(1M)
shl:	shell layer manager.	shl(1)
shutacct, startup, turnacct:	shell procedures for/ /runacct,	acctsh(1M)
File Sharing notification	shell script. /Remote	rfuadmin(1M)
command programming/ sh, rsh:	shell, the standard/restricted	sh(1)
	shl: shell layer manager.	shl(1)

system state. shutdown:	shut down system, change	shutdown(1M)
/prdaily, prtacct, runacct,	shutacct, startup, turnacct:/	acctsh(1M)
change system state.	shutdown: shut down system,	shutdown(1M)
program. sdiff:	side-by-side difference	sdiff(1)
login:	sign on.	login(1)
jwin: print	size of layer.	jwin(1)
an interval.	sleep: suspend execution for	sleep(1)
spline: interpolate	smooth curve.	spline(1G)
sort:	sort and/or merge files.	sort(1)
	sort: sort and/or merge files.	sort(1)
or reject lines common to two	sorted files. comm: select	comm(1)
idspace: investigates free	space.	idspace(1M)
terminal. ct:	spawn getty to a remote	ct(1C)
reads files containing	specifications. idmkinit:	idmkinit(1M)
/removes nodes and reads	specifications of nodes.	idmknod(1M)
/set terminal type, modes,	speed, and line discipline.	getty(1M)
/set terminal type, modes,	speed, and line discipline.	uugetty(1M)
hashcheck: find spelling/	spell, hashmake, spellin,	spell(1)
spelling/ spell, hashmake,	spellin, hashcheck: find	spell(1)
spellin, hashcheck: find	spelling errors. /hashmake,	spell(1)
curve.	spline: interpolate smooth	spline(1G)
split:	split a file into pieces.	split(1)
csplit: context	split.	csplit(1)
pieces.	split: split a file into	split(1)
uucleanup: uucp	spool directory clean-up.	uucleanup(1M)
lpadmin: configure the LP	spooling system.	lpadmin(1M)
sh, rsh: shell, the	standard/restricted command/	sh(1)
rfstart:	start Remote File Sharing.	rfstart(1M)
and/ lpsched, lpshut, lpmove:	start/stop the LP scheduler	lpsched(1M)
/prtacct, runacct, shutacct,	startup, turnacct: shell/	acctsh(1M)
ff: list file names and	statistics for a file system.	ff(1M)
extract and print xt driver	statistics. xts:	xts(1M)
fsstat: report file system	status.	fsstat(1M)
lpstat: print LP	status information.	lpstat(1)
control. uustat: uucp	status inquiry and job	uustat(1C)
communication facilities	status. /report interprocess	ipcs(1)
ps: report process	status.	ps(1)
rc0: run commands performed to	stop the operating system.	rc0(1M)
disk: random access bulk	storage medium.	disk(7)
messages.	strace: print STREAMS trace	strace(1M)
cleanup program.	strclean: STREAMS error logger	strclean(1M)
sed:	stream editor.	sed(1)
QIC cartridge magnetic tape	streamer interface. qt:	qt(7)
commands.	streamio: STREAMS ioctl	streamio(7)
open any minor device on a	STREAMS driver. clone:	clone(7)
program. strclean:	STREAMS error logger cleanup	strclean(1M)
strerr:	STREAMS error logger daemon.	strerr(1M)
event/ log: interface to	STREAMS error logging and	log(7)
streamio:	STREAMS ioctl commands.	streamio(7)
Interface cooperating	STREAMS module. /Transport	timod(7)
Interface read/write interface	STREAMS module. /Transport	tirdwr(7)
strace: print	STREAMS trace messages.	strace(1M)
daemon.	strerr: STREAMS error logger	strerr(1M)
search a file for a character	string. fgrep:	fgrep(1)
processes using a file or file	structure. fuser: identify	fuser(1M)
and print xt driver link	structure. xtd: extract	xtd(1M)
terminal.	stty: set the options for a	stty(1)
another user.	su: become super-user or	su(1M)
/same lines of several files or	subsequent lines of one file.	paste(1)

casual users). edit:	text editor (variant of ex for	edit(1)
change the format of a	text file. newform:	newform(1)
	tic: terminfo compiler.	tic(1M)
data and system/ timex:	time a command; report process	timex(1)
time:	time a command.	time(1)
execute commands at a later	time. at, batch:	at(1)
rtc: real	time clock interface.	rtc(7)
systems for optimal access	time. dcopy: copy file	dcopy(1M)
	time: time a command.	time(1)
update access and modification	times of a file. touch:	touch(1)
process data and system/	timex: time a command; report	timex(1)
cooperating STREAMS module.	timod: Transport Interface	timod(7)
read/write interface STREAMS/	tirdwr: Transport Interface	tirdwr(7)
tp4: Intel ISO TC4 compatible	TLI network device driver.	tp4(7)
acctmerg: merge or add	total accounting files.	acctmerg(1M)
modification times of a file.	touch: update access and	touch(1)
TLI network device driver.	tp4: Intel ISO TC4 compatible	tp4(7)
	tplot: graphics filters.	tplot(1G)
query terminfo data base.	tput: initialize a terminal or	tput(1)
	tr: translate characters.	tr(1)
strace: print STREAMS	trace messages.	strace(1M)
and print xt driver packet	traces. xtt: extract	xtt(1M)
error logging and event	tracing. /interface to STREAMS	log(7)
format: format floppy disk	tracks.	format(1M)
tr:	translate characters.	tr(1)
cooperating STREAMS/ timod:	Transport Interface	timod(7)
interface STREAMS/ tirdwr:	Transport Interface read/write	tirdwr(7)
system. uucico: file	transport program for the uucp	uucico(1M)
scheduler for the uucp file	transport program. /the	uusched(1M)
values.	true, false: provide truth	true(1)
i386: get processor type	truth value. machid:	machid(1)
true, false: provide	truth values.	true(1)
with debugging on. Uutry:	try to contact remote system	Uutry(1M)
interface.	tty: controlling terminal	tty(7)
terminals. xt: multiplexed	tty driver for AT&T windowing	xt(7)
terminal.	tty: get the name of the	tty(1)
attempts to set value of a	tunable parameter. idtune:	idtune(1M)
/runacct, shutacct, startup,	turnacct: shell procedures for/	acctsh(1M)
file: determine file	type.	file(1)
getty: set terminal	type, modes, speed, and line/	getty(1M)
uugetty: set terminal	type, modes, speed, and line/	uugetty(1M)
machid: i386: get processor	type truth value.	machid(1)
control.	uadmin: administrative	uadmin(1M)
mask.	umask: set file-creation mode	umask(1)
systems and remote/ mount,	umount: mount and unmount file	mount(1M)
multiple file/ mountall,	umountall: mount, unmount	mountall(1M)
File Sharing resource.	unadv: unadvertise a Remote	unadv(1M)
Sharing resource. unadv:	unadvertise a Remote File	unadv(1M)
UNIX system.	uname: print name of current	uname(1)
a file.	uniq: report repeated lines in	uniq(1)
	units: conversion program.	units(1)
link, unlink: link and	unlink files and directories.	link(1M)
and directories. link,	unlink: link and unlink files	link(1M)
mount, umount: mount and	unmount file systems and/	mount(1M)
mountall, umountall: mount,	unmount multiple file systems.	mountall(1M)
resource. fumount: forced	unmount of an advertised	fumount(1M)
rmountall, rumountall: mount,	unmount Remote File Sharing/	rmountall(1M)
files. pack, pcat,	unpack: compress and expand	pack(1)
times of a file. touch:	update access and modification	touch(1)

write:	write to another user.	write(1)
	write: write to another user.	write(1)
the 5620 DMD terminal.	wtinit: object downloader for	wtinit(1M)
accounting records. fwtmp,	wtmpfix: manipulate connect	fwtmp(1M)
list(s) and execute command.	xargs: construct argument	xargs(1)
xtd: extract and print	xt driver link structure.	xtd(1M)
xtt: extract and print	xt driver packet traces.	xtt(1M)
xts: extract and print	xt driver statistics.	xts(1M)
AT&T windowing terminals.	xt: multiplexed tty driver for	xt(7)
driver link structure.	xtd: extract and print xt	xtd(1M)
driver statistics.	xts: extract and print xt	xts(1M)
driver packet traces.	xtt: extract and print xt	xtt(1M)

Section 1

Section 1

NAME

intro – introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands (including system maintenance commands) available for your computer. The commands in this section should be used along with those listed in Sections 1, 2, 3, 4, and 5 of the *Programmer's Reference Manual.* References of the form *name*(1), *name*(2), *name*(3), *name*(4), and *name*(5) refer to entries in the above manual. References of the form *name*(1), *name*(1M), *name*(1C), *name*(1G), *name*(7), or *name*(8) refer to entries in this manual. Certain distinctions of purpose are made in the headings.

The following Utility packages are delivered with the computer:

Base System
Editing Package
Remote Terminal Package
Security Administration Package
2 Kilobyte File System Utility Package
Network Support Utilities Package
Remote File Sharing Utilities Package

Manual Page Command Syntax

Unless otherwise noted, commands described in the **SYNOPSIS** section of a manual page accept options and other arguments according to the following syntax and should be interpreted as explained below.

name [-*option*...] [*cmdarg*...]
where:

[ ]	Surround an *option* or *cmdarg* that is not required.
...	Indicates multiple occurrences of the *option* or *cmdarg*.
name	The name of an executable file.
option	(Always preceded by a "–".) *noargletter*... or, *argletter optarg*[,...]
noargletter	A single letter representing an option without an option-argument. Note that more than one *noargletter* option can be grouped after one "–" (Rule 5, in the following text).
argletter	A single letter representing an option requiring an option-argument.
optarg	An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas, or separated by white space and quoted (Rule 8, below).
cmdarg	Path name (or other command argument) *not* beginning with "–", or "–" by itself indicating the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands will obey them. *getopts*(1) should be used by all shell procedures to parse positional parameters and to check for legal options. It supports Rules 3-10 below. The enforcement of the other rules must be done by the command itself.

1. Command names (*name* above) must be between two and nine characters long.
2. Command names must include only lowercase letters and digits.
3. Option names (*option* above) must be one character long.
4. All options must be preceded by "-".
5. Options with no arguments may be grouped after a single "-".
6. The first option-argument (*optarg* above) following an option must be preceded by white space.
7. Option-arguments cannot be optional.
8. Groups of option-arguments following an option must either be separated by commas or separated by white space and quoted (e.g., `−o xxx,z,yy` or `−o "xxx z yy"`).
9. All options must precede operands (*cmdarg* above) on the command line.
10. "--" may be used to indicate the end of the options.
11. The order of the options relative to one another should not matter.
12. The relative order of the operands (*cmdarg* above) may affect their significance in ways determined by the command with which they appear.
13. "-" preceded and followed by white space should only be used to mean standard input.

SEE ALSO

getopts(1), exit(2).

wait(2), getopt(3C) in the *Programmer's Reference Manual.*

How to Get Started at the front of this document.

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 or bad or inaccessible data. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

BUGS

Regrettably, not all commands adhere to the aforementioned syntax.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

NAME

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

SYNOPSIS

300 [**+12**] [**-n**] [**-d**t,l,c]

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

DESCRIPTION

The *300* command 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. In the following discussion of the *300* command, it should be noted that unless your system contains the DOCUMENTER'S WORKBENCH Software, references to certain commands (e.g., *nroff, neqn, eqn,* etc.) will not work. 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%. The *300* command 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.

-d*t,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+(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 **nl0 cr2** or **nl0 cr3** are recommended for most uses.

The *300* command 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.

SEE ALSO

450(1), mesg(1), graph(1G), stty(1), tabs(1), tplot(1G).

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**] [**-c**N] [**-p**L] [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 Do not wait between pages (useful for directing output into a file).

-n Start printing at the current cursor position and never erase the screen.

-c*N* Divide the screen into *N* columns and wait after the last column.

-p*L* Set page length to *L*; *L* accepts the scale factors **i** (inches) and **l** (lines); default is lines.

SEE ALSO

pr(1).

NAME

450 – handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

The *450* command 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). It should be noted that, unless your system contains DOCUMENTER'S WORKBENCH Software, certain commands (e.g., *eqn*, *nroff*, *tbl*, etc.) will not work. Use *450* 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.

Use *450* 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.

SEE ALSO

300(1), mesg(1), stty(1), tabs(1), graph(1G), tplot(1G).

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

accept, reject – allow or prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [**-r**[reason]] destinations

DESCRIPTION

The *accept* command allows *lp*(1) to accept requests for the named *destinations*. A *destination* can be either a line printer (LP) or a class of printers. Use *lpstat*(1) to find the status of *destinations*.

The *reject* command prevents *lp*(1) from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*. The following option is useful with *reject*.

-r[*reason*] Associates a *reason* with preventing *lp* from accepting requests. This *reason* applies to all printers mentioned up to the next **-r** option. *Reason* is reported by *lp* when users direct requests to the named *destinations* and by *lpstat*(1). If the **-r** option is not present or the **-r** option is given without a *reason*, then a default *reason* will be used.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1).

NAME

acct: acctdisk, acctdusg, accton, acctwtmp – overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

/usr/lib/acct/acctdusg [**-u** file] [**-p** file]

/usr/lib/acct/accton [file]

/usr/lib/acct/acctwtmp "reason"

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. When the system is installed, accounting is initially in the "off" state. *acctsh*(1M) describes the set of shell procedures built on top of the C programs.

Connect time accounting is handled by various programs that write records into **/etc/utmp**, as described in *utmp*(4). The programs described in *acctcon*(1M) convert this file into session and charging records, which are then summarized by *acctmerg*(1M).

Process accounting is performed by the UNIX system kernel. Upon termination of a process, one record per process is written to a file (normally **/usr/adm/pacct**). The programs in *acctprc*(1M) summarize this data for charging purposes; *acctcms*(1M) is used to summarize command usage. Current process data may be examined using *acctcom*(1).

Process accounting and connect time accounting [or any accounting records in the format described in *acct*(4)] can be merged and summarized into total accounting records by *acctmerg* [see **tacct** format in *acct*(4)]. *prtacct* [see *acctsh*(1M)] is used to format any or all accounting records.

acctdisk reads lines that contain user ID, login name, and number of disk blocks and converts them to total accounting records that can be merged with other accounting records.

acctdusg reads its standard input (usually from **find / -print**) and computes disk resource consumption (including indirect blocks) by login. If **-u** is given, records consisting of those file names for which *acctdusg* charges no one are placed in *file* (a potential source for finding users trying to avoid disk charges). If **-p** is given, *file* is the name of the password file. This option is not needed if the password file is **/etc/passwd**. (See *diskusg*(1M) for more details.)

accton alone turns process accounting off. If *file* is given, it must be the name of an existing file, to which the kernel appends process accounting records [see *acct*(2) and *acct*(4)].

acctwtmp writes a *utmp*(4) record to its standard output. The record contains the current time and a string of characters that describe the *reason*. A record type of ACCOUNTING is assigned [see *utmp*(4)]. *Reason* must be a string of 11 or fewer characters, numbers, **$**, or spaces. For example, the following are suggestions for use in reboot and shutdown procedures, respectively:

```
acctwtmp uname >> /etc/wtmp
acctwtmp "file save" >> /etc/wtmp
```

FILES

/etc/passwd	used for login name to user ID conversions
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/pacct	current process accounting file
/etc/wtmp	login/logoff history file

SEE ALSO

acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M), runacct(1M).

acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

NAME

acctcms – command summary from per-process accounting records

SYNOPSIS

/usr/lib/acct/acctcms [options] files

DESCRIPTION

acctcms reads one or more *files*, normally in the form described in *acct*(4). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format. The *options* are:

-a Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, "hog factor", characters transferred, and blocks read and written, as in *acctcom*(1). Output is normally sorted by total kcore-minutes.

-c Sort by total CPU time, rather than total kcore-minutes.

-j Combine all commands invoked only once under "***other".

-n Sort by number of command invocations.

-s Any file names encountered hereafter are already in internal summary format.

-t Process all records as total accounting records. The default internal summary format splits each field into prime and non-prime time parts. This option combines the prime and non-prime time parts into a single field that is the total of both, and provides upward compatibility with old (i.e., UNIX System V) style **acctcms** internal summary format records.

The following options may be used only with the **-a** option.

-p Output a prime-time-only command summary.

-o Output a non-prime (offshift) time only command summary.

When **-p** and **-o** are used together, a combination prime and non-prime time report is produced. All the output summaries will be total usage except number of times executed, CPU minutes, and real minutes which will be split into prime and non-prime.

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms file ... >today
cp total previoustotal
acctcms –s today previoustotal >total
acctcms –a –s today
```

SEE ALSO

acct(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M).

acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

BUGS

Unpredictable output results if **-t** is used on new style internal summary format files, or if it is not used with old style internal summary format files.

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*), **STAT** (the system exit status), **HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD,** and **BLOCKS READ** (total blocks read and written).

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:

-a	Show some average statistics about the processes selected. The statistics will be printed after the output records.
-b	Read backwards, showing latest commands first. This *option* has no effect when the standard input is read.
-f	Print the *fork/exec* flag and system exit status columns in the output.
-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.
-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*. Using the same *time* for both **-S** and **-E** shows the processes that existed at *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.
-q	Do not print any output records, just print the average statistics as with the **-a** option.
-o *ofile*	Copy selected process records in the input data format to *ofile;* supress 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.
-I *chars*	Show only processes transferring more characters than the cut-off number given by *chars*.

FILES

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

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), ps(1), runacct(1M), su(1M).

acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

BUGS

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

NAME

acctcon: acctcon1, acctcon2 – connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/acctcon2

DESCRIPTION

acctcon1 converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from **/etc/wtmp**. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time. The *options* are:

-p Print input only, showing line name, login name, and time (in both numeric and date/time formats).

-t *acctcon1* maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The **-t** flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.

-l *file* *File* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Hang-up, termination of *login*(1) and termination of the login shell each generate logoff records, so that the number of logoffs is often three to four times the number of sessions. See *init*(1M) and *utmp*(4).

-o *file* *File* is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

acctcon2 expects as input a sequence of login session records and converts them into total accounting records [see **tacct** format in *acct*(4)].

EXAMPLES

These commands are typically used as shown below. The file **ctmp** is created only for the use of *acctprc*(1M) commands:

```
acctcon1 -t -l lineuse -o reboots <wtmp | sort +1n +2 >ctmp
acctcon2 <ctmp | acctmerg >ctacct
```

FILES

/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), init(1M), runacct(1M).
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

BUGS

The line usage report is confused by date changes. Use *wtmpfix* [see *fwtmp*(1M)] to correct this situation.

NAME

acctmerg – merge or add total accounting files

SYNOPSIS

/usr/lib/acct/acctmerg [options] [file] . . .

DESCRIPTION

acctmerg reads its standard input and up to nine additional files, all in the **tacct** format [see *acct*(4)] or an ASCII version thereof. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys. *Options* are:

-a	Produce output in ASCII version of **tacct**.
-i	Input files are in ASCII version of **tacct**.
-p	Print input with no processing.
-t	Produce a single record that totals all input.
-u	Summarize by user ID, rather than user ID and name.
-v	Produce output in verbose ASCII format, with more precise notation for floating point numbers.

EXAMPLES

The following sequence is useful for making "repairs" to any file kept in this format:

```
acctmerg -v <file1 >file2
        edit file2 as desired ...
acctmerg -i <file2 >file1
```

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M).

acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

NAME

acctprc: acctprc1, acctprc2 – process accounting

SYNOPSIS

/usr/lib/acct/acctprc1 [ctmp]

/usr/lib/acct/acctprc2

DESCRIPTION

acctprc1 reads input in the form described by *acct*(4), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in memory segment units). If **ctmp** is given, it is expected to contain a list of login sessions, in the form described in *acctcon*(1M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in **ctmp** helps it distinguish among different login names that share the same user ID.

acctprc2 reads records in the form written by *acctprc1*, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records.

These commands are typically used as shown below:

```
acctprc1 ctmp </usr/adm/pacct | acctprc2 >ptacct
```

FILES

/etc/passwd

SEE ALSO

acct(1M), acctcms(1M), acctcom(1) acctcon(1M), acctmerg(1M), acctsh(1M), cron(1M), fwtmp(1M), runacct(1M).

acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

BUGS

Although it is possible to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from *cron*(1M), for example. More precise conversion can be done by faking login sessions on the console via the *acctwtmp* program in *acct*(1M).

CAVEAT

A memory segment of the mean memory size is a unit of measure for the number of bytes in a logical memory segment on a particular processor. For example, on a PDP-11/70 this measure would be in 64-byte units, while on a VAX11/780 it would be in 512-byte units.

NAME

acctsh: chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct – shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee login-name number

/usr/lib/acct/ckpacct [blocks]

/usr/lib/acct/dodisk [-o] [files ...]

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct number

/usr/lib/acct/nulladm file

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [-l] [-c] [mmdd]

/usr/lib/acct/prtacct file ["heading"]

/usr/lib/acct/runacct [mmdd] [mmdd state]

/usr/lib/acct/shutacct ["reason"]

/usr/lib/acct/startup

/usr/lib/acct/turnacct on | off | switch

DESCRIPTION

chargefee can be invoked to charge a *number* of units to *login-name*. A record is written to **/usr/adm/fee**, to be merged with other accounting records during the night.

ckpacct should be initiated via **cron**(1M). It periodically checks the size of **/usr/adm/pacct**. If the size exceeds *blocks*, 1000 by default, **turnacct** will be invoked with argument *switch*. If the number of free disk blocks in the **/usr** file system falls below 500, **ckpacct** will automatically turn off the collection of process accounting records via the **off** argument to **turnacct**. When at least this number of blocks is restored, the accounting will be activated again. This feature is sensitive to the frequency at which **ckpacct** is executed, usually by **cron**.

dodisk should be invoked by **cron** to perform the disk accounting functions. By default, it will do disk accounting on the special files in **/etc/fstab.** If the **-o** flag is used, it will do a slower version of disk accounting by login directory. *Files* specify the one or more filesystem names where disk accounting will be done. If *files* are used, disk accounting will be done on these filesystems only. If the **-o** flag is used, *files* should be mount points of mounted filesystem. If omitted, they should be the special file names of mountable filesystems.

lastlogin is invoked by **runacct** to update **/usr/adm/acct/sum/loginlog**, which shows the last date on which each person logged in.

monacct should be invoked once each month or each accounting period. *Number* indicates which month or period it is. If *number* is not given, it defaults to the current month (01–12). This default is useful if **monacct** is to executed via **cron**(1M) on the first day of each month. **monacct** creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum**.

nulladm creates *file* with mode 664 and ensures that owner and group are **adm**. It is called by various accounting shell procedures.

prctmp can be used to print the session record file (normally **/usr/adm/acct/nite/ctmp** created by **acctcon**(1M).

prdaily is invoked by **runacct** to format a report of the previous day's accounting data. The report resides in **/usr/adm/acct/sum/rprt***mmdd* where *mmdd* is the month and day of the report. The current daily accounting reports may be printed by typing **prdaily**. Previous days' accounting reports can be printed by using the *mmdd* option and specifying the exact report date desired. The **-l** flag prints a report of exceptional usage by login id for the specifed date. Previous daily reports are cleaned up and therefore inaccessible after each invocation of **monacct**. The **-c** flag prints a report of exceptional resource usage by command, and may be used on current day's accounting data only.

prtacct can be used to format and print any total accounting (**tacct**) file.

runacct performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see **runacct**(1M).

shutacct is invoked during a system shutdown to turn process accounting off and append a "reason" record to **/etc/wtmp**.

startup is called by **/etc/init.d/acct** to turn the accounting on whenever the system is brought to a multi-user state.

turnacct is an interface to **accton** [see **acct**(1M)] to turn process accounting **on** or **off**. The **switch** argument turns accounting off, moves the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct***incr* (where *incr* is a number starting with **1** and incrementing by one for each additional **pacct** file), then turns accounting back on again. This procedure is called by **ckpacct** and thus can be taken care of by the **cron** and used to keep **pacct** to a reasonable size. **acct** starts and stops process accounting via **init** and **shutdown** accordingly.

FILES

/usr/adm/fee	accumulator for fees
/usr/adm/pacct	current file for per-process accounting
/usr/adm/pacct*	used if pacct gets large and during execution of daily accounting procedure
/etc/wtmp	login/logoff summary
/usr/lib/acct/ptelus.awk	contains the limits for exceptional usage by login id
/usr/lib/acct/ptecms.awk	contains the limits for exceptional usage by command name
/usr/adm/acct/nite	working directory

/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/acct/sum	summary directory, should be saved

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), cron(1M), diskusg(1M), fwtmp(1M), runacct(1M).

acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

NAME

adm – invoke the AT&T Administration interface

SYNOPSIS

adm

DESCRIPTION

The *adm* command is used to invoke the forms-and-menus interface for administering the computer. It is designed to make administration easy enough for the average user to handle, with sufficient help to make each step self-explanatory. Using this interface, you can:

Administer user logins*

Set up peripheral devices*

Change the date and time*

Shut down the computer to turn it off*

Administer an attached printer

Format and copy floppy diskettes

Create, mount*, and unmount* file systems

Backup and Restore files from the hard disk*

Set up your system for electronic mail*

Report system configuration information

For additional information, see the *Operations/System Administration Guide*. Items marked with an asterisk require system administration privileges, which can be given to any user by *root*, or by another user with such privileges.

The interface is designed to keep the administration files of the computer self-consistent. Making changes to system files with an editor, outside the interface, could create inconsistencies that the interface is not prepared to handle, and could result in a non-functional system.

The interface looks at the TERM environment variable to determine what *terminfo* entry to use. Basic navigation is accomplished with arrow keys to move, the return/enter key to select, and function keys as labeled for other tasks. To refresh the screen in the event of unexpected error messages (such as from disk bad-block handling), type control-Z followed by "refresh" and a return/enter.

NAME

adv – advertise a directory for remote access

SYNOPSIS

adv [**-r**] [**-d** *description*] *resource pathname* [*clients*...]

adv -m *resource* **-d** *description* ¦ [*clients*...]

adv -m *resource* [**-d** *description*] ¦ *clients*...

adv

DESCRIPTION

The **adv** command is the Remote File Sharing command used to make a computer's resource available to other computers. The machine that advertises the resource is called the *server,* while computers that mount and use the resource are *clients.* [See **mount**(1M).] (A resource represents a directory, which could contain files, subdirectories, named pipes and devices.)

There are three ways **adv** is used: 1. to advertise the directory *pathname* under the name *resource* so it is available to Remote File Sharing *clients;* 2. to modify *client* and *description* fields for currently advertised resources; or 3. to print a list of all locally advertised resources.

The following options are available:

-r	Restricts access to the resource to a read-only basis. The default is read-write access.
-d *description*	Provides brief textual information about the advertised resource. *description* is a single argument surrounded by double quotes (") and has a maximum length of 32 characters.
resource	This is the symbolic name used by the server and all authorized clients to identify the resource. It is limited to a maximum of 14 characters and must be different from every other resource name in the domain. All characters must be printable ASCII characters but must not include periods (.), slashes (/), or white space.
pathname	This is the local path name of the advertised resource. It is limited to a maximum of 64 characters. This path name cannot be the mount point of a remote resource and it can only be advertised under one resource name.
clients	These are the names of all clients that are authorized to remotely mount the resource. The default is that all machines that can connect to the server are authorized to access the resource. Valid input is of the form *nodename, domain.nodename, domain.,* or an alias that represents a list of client names. A domain name must be followed by a period (.) to distinguish it from a host name. The aliases are defined in **/etc/host.alias** and must conform to the alias capability in **mailx**(1).

-m *resource*	This option modifies information for a resource that has already been advertised. The resource is identified by a *resource* name. Only the *clients* and *description* fields can be modified. (To change the *pathname, resource* name, or read/write permissions, you must unadvertise and re-advertise the resource.)

When used with no options, *adv* displays all local resources that have been advertised; this includes the resource name, the path name, the description, the read-write status, and the list of authorized clients. The resource field has a fixed length of 14 characters; all others are of variable length. Fields are separated by two white spaces, double quotes (") surround the description, and blank lines separate each resource entry.

This command may be used without options by any user; otherwise it is restricted to the super-user.

Remote File Sharing must be running before **adv** can be used to advertise or modify a resource entry.

EXIT STATUS

If there is at least one syntactically valid entry in the *clients* field, a warning will be issued for each invalid entry and the command will return a successful exit status. A non-zero exit status will be returned if the command fails.

ERRORS

If (1) the network is not up and running, (2) *pathname* is not a directory, (3) *pathname* isn't on a file system mounted locally, or (4) there is at least one entry in the *clients* field but none are syntactically valid, an error message will be sent to standard error.

FILES

/etc/host.alias

SEE ALSO

mailx(1) mount(1M), rfstart(1M), unadv(1M).

NAME

at, batch – execute commands at a later time

SYNOPSIS

at *time* [*date*] [+ *increment*]
at -r job ...
at -l [*job*

batch

DESCRIPTION

The *at* and *batch* commands read commands from standard input to be executed at a later time. *at* allows you to specify when the commands should be executed, while jobs queued with *batch* will execute when system load level permits. *at* may be used with the following options:

-r Removes jobs previously scheduled with *at*.

-l Reports all jobs scheduled for the invoking user.

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

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

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

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

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

Thus legitimate commands include:

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

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

The *at* **-r** command removes jobs previously scheduled by *at* or *batch*. The job number is the number given to you previously by the *at* or *batch* command. You can also get job numbers by typing *at* **-l**. You can remove only your own jobs unless you are the super-user.

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *sh*(1) provides a different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch
sort filename >outfile
<control-D> (hold down 'control' and depress 'D')
```

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

```
batch <<!
sort filename 2>&1 >outfile | mail loginid
!
```

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

```
echo "sh shellfile" | at 1900 thursday next week
```

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/at.allow	list of allowed users
/usr/lib/cron/at.deny	list of denied users
/usr/lib/cron/queue	scheduling information
/usr/spool/cron/atjobs	spool area

SEE ALSO

cron(1M), kill(1), mail(1), nice(1), ps(1), sh(1), sort(1).

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk – pattern scanning and processing language

SYNOPSIS

awk [**-Fc**] [prog] [parameters] [files]

DESCRIPTION

The *awk* language 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) in the *Programmer's Reference Manual*].

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp, log, sqrt,* and *int*. 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 **-F***c* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default 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), sed(1).
lex(1), printf(3S) in the *Programmer's Reference Manual.*

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

backup – performs backup functions

SYNOPSIS

backup [-**t**] [-**p** | -**c** | -**f** <files> | -**u** "<user1> [user2]"]
  -**d** <device>

backup -h

DESCRIPTION

-h produces a history of backups. Tells the user when the last complete and incremental/partial backups were done.

-c complete backup. All files changed since the system was installed are backed up.

-p incremental/partial backup. If a incremental/partial backup was done, all files modified since that time are backed up, otherwise all files modified since the last complete backup are backed up. A complete backup must be done before a partial backup.

-f backup files specified by the *<files>* argument. File names may contain characters to be expanded (i.e., *, .) by the shell. The argument must be in quotes.

-u backup a users files. At least one user must be specified but it can be more. The argument must be in quotes if more than one user is specified. User name of "all" causes all users to be backed up. All the files belonging to the specified users will be backed up.

-d used to specify the device to be used. It defaults to **/dev/rdsk/f0q15d** (the 1.2M floppy).

-t used when the device is the tape. It must be used with the **-d** option.

A complete backup must be done before a partial backup can be done.

Raw devices rather than block devices should always be used.

The program can handle multi-volume backups.

The program will prompt the user when it is ready for the next media.

The program will give you an estimated number of floppies/tapes that will be needed to do the backup.

Floppies MUST be formatted before the backup is done.

Tapes do not need to be formatted.

If backup is done to tape, the tape must be rewound.

SEE ALSO

qt(7).

NAME

banner – make posters

SYNOPSIS

banner strings

DESCRIPTION

The *banner* command prints its arguments (each up to 10 characters long) in large letters on the standard output. Spaces can be included in an argument by surrounding it with quotes. The maximum number of characters that can be accommodated in a line is implementation-dependent; excess characters are simply ignored.

SEE ALSO

echo(1).

NAME

basename, dirname – deliver portions of path names

SYNOPSIS

basename string [suffix]
dirname string

DESCRIPTION

The *basename* command 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.

The *dirname* command 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).

NAME

bc – arbitrary-precision arithmetic language

SYNOPSIS

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

DESCRIPTION

The *bc* command 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 *bc*(1) utility is actually a preprocessor for *dc*(1), which it invokes automatically unless the **-c** option is present. In this case the *dc* input is sent to the standard output instead. The options are as follows:

-c Compile only. The output is send to the standard output.

-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.

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).

BUGS

The *bc* command does not yet recognize the logical operators, **&&** and **| |**.
For statement must have all three expressions (E's).
Quit is interpreted when read, not when executed.

NAME

bdiff – big diff

SYNOPSIS

bdiff file1 file2 [n] [**-s**]

DESCRIPTION

The *bdiff* command is used in a manner analogous to *diff*(1) to find which lines in two files must be changed to bring the files into agreement. Its purpose is to allow processing of files which are too large for *diff*.

The parameters to *bdiff* are:

file1 (file2)
: The name of a file to be used. If *file1 (file2)* is –, the standard input is read.

n
: The number of line 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.

-s
: Specifies that no diagnostics are to be printed by *bdiff* (silent option). Note, however, that this does not suppress possible diagnostic messages from *diff*(1), which *bdiff* calls.

The *bdiff* command 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. 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

The *bfs* command is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes and 32K lines, with up to 512 characters, including new-line, per line (255 for 16-bit machines). *bfs* is usually more efficient than *ed*(1) 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*(1). 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*(1) 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. 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**, **p**, **q**, **w**, **=**, **!** and null commands operate as described under *ed*(1). 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 **xo**, **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**. The **xf** commands may be nested to a depth of 10.

xn
: List the marks currently in use (marks are set by the **k** command).

xo [*file*]
: Further output from the **p** and null commands is diverted to the named *file*, which, if necessary, is created mode 666 (readable and writable by everyone), unless your *umask* setting [see *umask*(1)] dictates otherwise. 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 does not 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 does not 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**. The commands **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. The command **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 system 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 system 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 is not. 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), umask(1).

DIAGNOSTICS

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

NAME

brc, bcheckrc – system initialization procedures

SYNOPSIS

/etc/brc

/etc/bcheckrc

DESCRIPTION

These shell procedures are executed via entries in **/etc/inittab** by *init*(1M) whenever the system is booted (or rebooted).

First, the *bcheckrc* procedure checks the status of the root file system. If the root file system is found to be bad, *bcheckrc* repairs it.

Then, the *brc* procedure clears the mounted file system table, **/etc/mnttab** and puts the entry for the root file system into the mount table.

After these two procedures have executed, *init* checks for the *initdefault* value in **/etc/inittab**. This tells *init* in which run level to place the system. Since *initdefault* is initially set to **2**, the system will be placed in the multi-user state via the */etc/rc2* procedure.

Note that *bcheckrc* should always be executed before *brc*. Also, these shell procedures may be used for several run-level states.

SEE ALSO

fsck(1M), init(1M), rc2(1M), shutdown(1M).

NAME

cal – print calendar

SYNOPSIS

cal [[month] year]

DESCRIPTION

The *cal* command prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present 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 the United States.

EXAMPLES

An unusual calendar is printed for September 1752. That is the month 11 days were skipped to make up for lack of leap year adjustments. To see this calendar, type: **cal 9 1752**

BUGS

The year is always considered to start in January even though this is historically naive.
Beware that "cal 83" refers to the early Christian era, not the 20th century.

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

The *calendar* command 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 "Aug. 24," "august 24," "8/24," etc., are recognized, but not "24 August" or "24/8". On weekends "tomorrow" extends through Monday.

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

FILES

/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

captoinfo – convert a termcap description into a terminfo description

SYNOPSIS

captoinfo [**-v** ...] [**-V**] [**-1**] [**-w** width] file ...

DESCRIPTION

The *captoinfo* command looks in *file* for *termcap* descriptions. For each one found, an equivalent *terminfo*(4) description is written to standard output, along with any comments found. A description which is expressed as relative to another description (as specified in the *termcap tc=* field) will be reduced to the minimum superset before being output.

If no *file* is given, then the environment variable **TERMCAP** is used for the file name or entry. If **TERMCAP** is a full path name to a file, only the terminal whose name is specified in the environment variable **TERM** is extracted from that file. If the environment variable **TERMCAP** is not set, then the file */etc/termcap* is read.

-v print out tracing information on standard error as the program runs. Specifying additional **-v** options will cause more detailed information to be printed.

-V print out the version of the program in use on standard error and exit.

-1 cause the fields to print out, one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

-w change the output to *width* characters.

FILES

/usr/lib/terminfo/?/* compiled terminal description data base

CAVEATS

Certain *termcap* defaults are assumed to be true. For example, the bell character (*terminfo bel*) is assumed to be ^*G*. The linefeed capability (*termcap nl*) is assumed to be the same for both *cursor_down* and *scroll_forward* (*terminfo cud1* and *ind*, respectively.) Padding information is assumed to belong at the end of the string.

The algorithm used to expand parameterized information for *termcap* fields such as *cursor_position* (*termcap cm, terminfo cup*) will sometimes produce a string which, though technically correct, may not be optimal. In particular, the rarely used *termcap* operation **%n** will produce strings that are especially long. Most occurrences of these non-optimal strings will be flagged with a warning message and may need to be recoded by hand.

The short two-letter name at the beginning of the list of names in a *termcap* entry, a hold-over from an earlier version of the UNIX system, has been removed.

DIAGNOSTICS

tgetent failed with return code n (reason).
: The termcap entry is not valid. In particular, check for an invalid 'tc=' entry.

unknown type given for the termcap code *cc*.
: The termcap description had an entry for *cc* whose type was not Boolean, numeric, or string.

wrong type given for the Boolean (numeric, string) termcap code *cc*.
: The Boolean *termcap* entry *cc* was entered as a numeric or string capability.

the Boolean (numeric, string) termcap code *cc* is not a valid name.
: An unknown *termcap* code was specified.

tgetent failed on TERM=term.
: The terminal type specified could not be found in the *termcap* file.

TERM=term: **cap** *cc* (**info** *ii*) is NULL: REMOVED
: The *termcap* code was specified as a null string. The correct way to cancel an entry is with an '@', as in ':bs@:'. Giving a null string could cause incorrect assumptions to be made by the software which uses *termcap* or *terminfo*.

a function key for *cc* was specified, but it already has the value *vv*.
: When parsing the **ko** capability, the key *cc* was specified as having the same value as the capability *cc*, but the key *cc* already had a value assigned to it.

the unknown termcap name *cc* was specified in the **ko** termcap capability.
: A key was specified in the **ko** capability which could not be handled.

the *vi* character *v* (**info** *ii*) has the value *xx*, but **ma** gives *n*.
: The **ma** capability specified a function key with a value different from that specified in another setting of the same key.

the unknown *vi* key *v* was specified in the **ma** termcap capability.
: A *vi*(1) key unknown to *captoinfo* was specified in the **ma** capability.

Warning: *termcap* **sg** (*nn*) and *termcap* **ug** (*nn*) had different values.
: *terminfo* assumes that the **sg** (now **xmc**) and **ug** values were the same.

Warning: the string produced for *ii* may be inefficient.
: The parameterized string being created should be rewritten by hand.

Null termname given.
The terminal type was null. This is given if the environment variable **TERM** is not set or is null.

cannot open *file* for reading.
The specified file could not be opened.

SEE ALSO

infocmp(1M), tic(1M).
curses (3X), terminfo(4) in the *Programmer's Reference Manual.*
Chapter 10 in the *Programmer's Guide*.

NOTES

The *captoinfo* command should be used to convert *termcap* entries to *terminfo*(4) entries because the *termcap* data base (from earlier versions of UNIX System V) may not be supplied in future releases.

NAME

cat – concatenate and print files

SYNOPSIS

cat [**-u**] [**-s**] [**-v** [**-t**] [**-e**]] file ...

DESCRIPTION

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

cat file

prints **file** on your terminal, and:

cat file1 file2 >file3

concatenates **file1** and **file2**, and writes the results in **file3**.

If no input file is given, or if the argument - is encountered, *cat* reads from the standard input file.

The following options apply to *cat*:

-u The output is not buffered. (The default is buffered output.)

-s *cat* is silent about non-existent files.

-v Causes non-printing characters (with the exception of tabs, new-lines, and form-feeds) to be printed visibly. ASCII control characters (octal 000 - 037) are printed as ^*n*, where *n* is the corresponding ASCII character in the range octal 100 - 137 (@, A, B, C, . . ., X, Y, Z, [, \,], ^, and _); the DEL character (octal 0177) is printed ^**?**. Other non-printable characters are printed as **M-***x*, where *x* is the ASCII character specified by the low-order seven bits.

When used with the **-v** option, the following options may be used:

-t Causes tabs to be printed as ^**I**'s.

-e Causes a **$** character to be printed at the end of each line (prior to the new-line).

The **-t** and **-e** options are ignored if the **-v** option is not specified.

WARNING

Redirecting the output of **cat** onto one of the files being read will overwrite the data originally in the file being read. For example, typing:

cat file1 file2 >file1

will cause the original data in **file1** to be lost.

SEE ALSO

cp(1), pg(1), pr(1).

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 is internal to the shell.

SEE ALSO

pwd(1), sh(1).
chdir(2) in the *Programmer's Reference Manual.*

NAME

chmod – change mode

SYNOPSIS

chmod mode file ...

chmod mode directory ...

DESCRIPTION

The permissions of the named *files* or *directories* are **ch**anged according to **mode**, which may be symbolic or absolute. Absolute changes to permissions are stated using octal numbers:

chmod *nnn file(s)*

where *n* is a number from 0 to 7. Symbolic changes are stated using mnemonic characters:

chmod *a operator b file(s)*

where *a* is one or more characters corresponding to **user**, **group**, or **other**; where *operator* is +, –, and =, signifying assignment of permissions; and where *b* is one or more characters corresponding to type of permission.

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

4000	set user ID on execution
20#0	set group ID on execution if # is **7**, **5**, **3**, or **1**
	enable mandatory locking if # is **6**, **4**, **2**, or **0**
1000	sticky bit is turned on [see *chmod*(2)]
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

Symbolic changes are stated using letters that correspond both to access classes and to the individual permissions themselves. Permissions to a file may vary depending on your user identification number (UID) or group identification number (GID). Permissions are described in three sequences each having three characters:

User	Group	Other
rwx	rwx	rwx

This example (meaning that **u**ser, **g**roup, and **o**thers all have **r**eading, **w**riting, and e**x**ecution permission to a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

Thus, to change the mode of a file's (or directory's) permissions using *chmod*'s symbolic method, use the following syntax for mode:

[*who*] *operator* [*permission(s)*], ...

A command line using the symbolic method would appear as follows:

chmod g+rw *file*

This command would make *file* readable and writable by the group.

The *who* part can be stated as one or more of the following letters:

u	**u**ser's permissions
g	**g**roup's permissions
o	**o**thers permissions

The letter **a** (all) is equivalent to **ugo** and is the default if *who* is omitted.

Operator can be + to add *permission* to the file's mode, - to take away *permission*, or = to assign *permission* absolutely. (Unlike other symbolic operations, = has an absolute effect in that it resets all other bits.) Omitting *permission* is only useful with = to take away all permissions.

Permission is any compatible combination of the following letters:

r	**r**eading permission
w	**w**riting permission
x	e**x**ecution permission
s	user or group **s**et-ID is turned on
t	s**t**icky bit is turned on
l	mandatory **l**ocking will occur during access

Multiple symbolic modes separated by commas may be given, though no spaces may intervene between these modes. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously. The letter **s** is only meaningful with **u** or **g**, and **t** only works with **u**.

Mandatory file and record locking (**l**) refers to a file's ability to have its reading or writing permissions locked while a program is accessing that file. It is not possible to permit group execution and enable a file to be locked on execution at the same time. In addition, it is not possible to turn on the set-group-ID and enable a file to be locked on execution at the same time. The following examples,

chmod g+x,+l *file*
chmod g+s,+l *file*

are, therefore, illegal usages and will elicit error messages.

Only the owner of a file or directory (or the super-user) may change a file's mode. Only the super-user may set the sticky bit on a non-directory file. In order to turn on a file's set-group-ID, your own group ID must correspond to the file's, and group execution must be set.

EXAMPLES

chmod a–x *file*

chmod 444 *file*

The first examples deny execution permission to all. The absolute (octal) example permits only reading permissions.

chmod go+rw *file*

chmod 606 *file*

These examples make a file readable and writable by the group and others.

```
chmod +l file
```

This causes a file to be locked during access.

```
chmod =rwx,g+s file

chmod 2777 file
```

These last two examples enable all to read, write, and execute the file; and they turn on the set group-ID.

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls -l** command leads you to believe. For more information see the "Mapping Remote Users" section of Chapter 10 of the *System Administrator's Guide*.

SEE ALSO

ls(1).
chmod(2) in the *Programmer's Reference Manual*.

NAME

chown, chgrp – change owner or group

SYNOPSIS

chown owner file ...

chown owner directory ...

chgrp group file ...

chgrp group directory ...

DESCRIPTION

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

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

If either command 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.

Only the owner of a file (or the super-user) may change the owner or group of that file.

FILES

/etc/passwd
/etc/group

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls -l** command leads you to believe. For more information see the "Mapping Remote Users" section of Chapter 10 of the *System Administrator's Guide.*

SEE ALSO

chmod(1).
chown(2), group(4), passwd(4) in the *Programmer's Reference Manual.*

NAME

chroot – change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

The *chroot* command causes the given command to be executed relative to the new root. The meaning of any initial slashes (/) in the path names is changed for the command and any of its child processes to *newroot*. Furthermore, upon execution, the initial working directory is *newroot*.

Notice, however, that if you redirect the output of the command to a file:

```
chroot newroot command >x
```

will create the file **x** relative to the original root of the command, not the new one.

The new root path name is always relative to the current root; even if a *chroot* is currently in effect, the *newroot* argument is relative to the current root of the running process.

This command can be run only by the super-user.

SEE ALSO

cd(1).
chroot(2) in the *Programmer's Reference Manual.*

BUGS

One should exercise extreme caution when referencing device files in the new root file system.

NAME

chrtbl – generate character classification and conversion tables

SYNOPSIS

chrtbl [file]

DESCRIPTION

The *chrtbl* command creates a character classification table and an upper/lower-case conversion table. The tables are contained in a byte-sized array encoded such that a table lookup can be used to determine the character classification of a character or to convert a character [see *ctype*(3C)]. The size of the array is 257*2 bytes: 257 bytes are required for the 8-bit code set character classification table and 257 bytes for the upper- to lower-case and lower- to upper-case conversion table.

chrtbl reads the user-defined character classification and conversion information from *file* and creates two output files in the current directory. One output file, **ctype.c** (a C-language source file), contains the 257*2-byte array generated from processing the information from *file*. You should review the content of **ctype.c** to verify that the array is set up as you had planned. (In addition, an application program could use **ctype.c**.) The first 257 bytes of the array in **ctype.c** are used for character classification. The characters used for initializing these bytes of the array represent character classifications that are defined in **/usr/include/ctype.h**; for example, **_L** means a character is lower case and **_S | _B** means the character is both a spacing character and a blank. The last 257 bytes of the array are used for character conversion. These bytes of the array are initialized so that characters for which you do not provide conversion information will be converted to themselves. When you do provide conversion information, the first value of the pair is stored where the second one would be stored normally, and vice versa; for example, if you provide <**0x41 0x61**>, then **0x61** is stored where **0x41** would be stored normally, and **0x61** is stored where **0x41** would be stored normally.

The second output file (a data file) contains the same information, but is structured for efficient use by the character classification and conversion routines [see *ctype*(3C)]. The name of this output file is the value of the character classification **chrclass** read in from *file*. This output file must be installed in the **/lib/chrclass** directory under this name by someone who is super-user or a member of group **bin**. This file must be readable by user, group, and other; no other permissions should be set. To use the character classification and conversion tables on this file, set the environmental variable **CHRCLASS** [see *environ*(5)] to the name of this file and export the variable; for example, if the name of this file (and character class) is **xyz**, you should issue the commands: **CHRCLASS=xyz ; export CHRCLASS .**

If no input file is given, or if the argument – is encountered, *chrtbl* reads from the standard input file.

The syntax of *file* allows the user to define the name of the data file created by *chrtbl*, the assignment of characters to character classifications and the relationship between upper- and lower-case letters. The character classifications recognized by *chrtbl* are:

chrclass	name of the data file to be created by *chrtbl*.
isupper	character codes to be classified as upper-case letters.
islower	character codes to be classified as lower-case letters.
isdigit	character codes to be classified as numeric.
isspace	character codes to be classified as a spacing (delimiter) character.
ispunct	character codes to be classified as a punctuation character.
iscntrl	character codes to be classified as a control character.
isblank	character code for the space character.
isxdigit	character codes to be classified as hexadecimal digits.
ul	relationship between upper- and lower-case characters.

Any lines with the number sign (#) in the first column are treated as comments and are ignored. Blank lines are also ignored.

A character can be represented as a hexadecimal or octal constant (for example, the letter **a** can be represented as 0x61 in hexadecimal or 0141 in octal). Hexadecimal and octal constants may be separated by one or more space and tab characters.

The dash character (-) may be used to indicate a range of consecutive numbers. Zero or more space characters may be used for separating the dash character from the numbers.

The backslash character (\) is used for line continuation. Only a carriage return is permitted after the backslash character.

The relationship between upper- and lower-case letters (**ul**) is expressed as ordered pairs of octal or hexadecimal constants: <*upper-case_character lower-case_character*>. These two constants may be separated by one or more space characters. Zero or more space characters may be used for separating the angle brackets (< >) from the numbers.

EXAMPLE

The following is an example of an input file used to create the ASCII code set definition table on a file named **ascii**.

```
chrclass  ascii
isupper   0x41 - 0x5a
islower   0x61 - 0x7a
isdigit   0x30 - 0x39
isspace   0x20 0x9 - 0xd
ispunct   0x21 - 0x2f 0x3a - 0x40    \
          0x5b - 0x60 0x7b - 0x7e
iscntrl   0x0 - 0x1f  0x7f
isblank   0x20
isxdigit  0x30 - 0x39 0x61 - 0x66    \
                 0x41 - 0x46
        ul      <0x41 0x61> <0x42 0x62> <0x43 0x63>  \
                <0x44 0x64> <0x45 0x65> <0x46 0x66>  \
                <0x47 0x67> <0x48 0x68> <0x49 0x69>  \
                <0x4a 0x6a> <0x4b 0x6b> <0x4c 0x6c>  \
                <0x4d 0x6d> <0x4e 0x6e> <0x4f 0x6f>  \
                <0x50 0x70> <0x51 0x71> <0x52 0x72>  \
                <0x53 0x73> <0x54 0x74> <0x55 0x75>  \
                <0x56 0x76> <0x57 0x77> <0x58 0x78>  \
                <0x59 0x79> <0x5a 0x7a>
```

FILES

/lib/chrclass/*	data file containing character classification and conversion tables created by *chrtbl*
/usr/include/ctype.h	header file containing information used by character classification and conversion routines

SEE ALSO

environ(5).
ctype(3C) in the *Programmer's Reference Manual* .

DIAGNOSTICS

The error messages produced by *chrtbl* are intended to be self-explanatory. They indicate errors in the command line or syntactic errors encountered within the input file.

NAME

clri – clear i-node

SYNOPSIS

/etc/clri special i-number ...

DESCRIPTION

The *clri* command writes nulls on the 64 bytes at offset *i-number* from the start of the i-node list. This effectively eliminates the i-node at that address. *Special* is the device name on which a file system has been defined. After *clri* is executed, any blocks in the affected file will show up as "not accounted for" when *fsck*(1M) is run against the file-system. The i-node may be allocated to a new file.

Read and write permission is required on the specified *special* device.

This command is used to remove a file which appears in no directory; that is, to get rid of a file which cannot be removed with the *rm* command.

SEE ALSO

fsck(1M), fsdb(1M), ncheck(1M) rm(1).
fs(4) in the *Programmer's Reference Manual.*

WARNINGS

If the file is open for writing, *clri* will not work. The file system containing the file should NOT be mounted.

If *clri* is used on the i-node number of a file that does appear in a directory, it is imperative to remove the entry in the directory at once, since the i-node may be allocated to a new file. The old directory entry, if not removed, continues to point to the same file. This sounds like a link, but does not work like one. Removing the old entry destroys the new file.

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**] [**-f**] [**-x**] [**-p**]

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 escape sequences unknown to it that are 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.

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

comm – select or reject lines common to two sorted files

SYNOPSIS

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

DESCRIPTION

The *comm* command 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** prints nothing.

SEE ALSO

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

NAME

cp, ln, mv – copy, link, or move files

SYNOPSIS

cp file1 [file2 ...] target
ln [**-f**] file1 [file2 ...] target
mv [**-f**] 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 *target* is a file, its contents are destroyed.

If *mv* or *ln* determines that the mode of *target* forbids writing, it will print the mode [see *chmod*(2)], ask for a response, and read the standard input for one line; if the line begins with **y**, the *mv* or *ln* occurs, if permissable; if not, the command exits. For *mv*, when source parent directories or the target directory is writable and has the sticky bit set, any of the following conditions must be true:

> the user must own the file
> the user must own the directory
> the file must be writable to the user
> the user must be the super-user

When the **-f** option is used or if the standard input is not a terminal, no questions are asked and the *mv* or *ln* is done.

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; *file1* is renamed *target*. If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

When using *cp*, if *target* is not a file, a new file is created which has the same mode as *file1* except that the sticky bit is not set unless you are super-user; the owner and group of *target* are those of the user. If *target* is a file, copying a file into *target* does not change its mode, owner, nor group. The last modification time of *target* (and last access time, if *target* did not exist) and the last access time of *file1* are set to the time the copy was made. If *target* is a link to a file, all links remain and the file is changed.

SEE ALSO

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

WARNINGS

ln will not link across file systems. This restriction is necessary because file systems can be added and removed.

BUGS

If *file1* and *target* lie on different file systems, *mv* must copy the file and delete the original. In this case any linking relationship with other files is lost.

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio -o[**acBvV**] [**-C** *bufsize*] [[**-O** *file*] [**-M** *message*]]

cpio -i[**BcdmrtuvVfsSb6k**] [**-C** *bufsize*] [[**-I** *file*] [**-M** *message*]] [*pattern* ...]

cpio -p[**adlmuvV**] 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. Output is padded to a 512-byte boundary by default.

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 regular expressions given in the filename-generating notation of *sh*(1). In *patterns*, meta-characters **?**, *, and **[...]** match the slash (**/**) character, and backslash (\) is an escape character. A **!** meta-character means *not.* (For example, the **!abc*** pattern would exclude all files that begin with **abc**.) Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). Each *pattern* must be enclosed in double quotes otherwise the name of a file in the current directory is used. Extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous **cpio -o**. The owner and group of the files will be that of the current user unless the user is super-user, which causes *cpio* to retain the owner and group of the files of the previous **cpio -o**. NOTE: If **cpio -i** tries to create a file that already exists and the existing file is the same age or newer, *cpio* will output a warning message and not replace the file. (The **-u** option can be used to unconditionally overwrite the existing file.)

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. Archives of text files created by **cpio** are portable between implementations of System V.

The meanings of the available options are

-a Reset *access* times of input files after they have been copied. Access times are not reset for linked files when **cpio -pla** is specified.

-b Reverse the order of the *bytes* within each word. Use only with the **-i** option.

-B Input/output is to be blocked 5,120 bytes to the record. The default buffer size is 512 bytes when this and the **C** options are not used. (**-B** does not apply to the *pass* option; **-B** is meaningful only with data directed to or from a character-special device, e.g. **/dev/rdsk/f0q15dt**.)

-c Write header information in ASCII *character* form for portability. Always use this option when origin and destination machines are different types.

-C *bufsize*
: Input/output is to be blocked *bufsize* bytes to the record, where *bufsize* is replaced by a positive integer. The default buffer size is 512 bytes when this and **B** options are not used. (**-C** does not apply to the *pass* option; **-C** is meaningful only with data directed to or from a character special device, e.g. **/dev/rmt/c0s0**.)

-d *directories* are to be created as needed.

-f Copy in all *files* except those in *patterns.* (See the paragraph on **cpio -i** for a description of *patterns.*)

-I *file* Read the contents of *file* as input. If *file* is a character special device, when the first medium is full replace the medium and type a carriage return to continue to the next medium. Use only with the **-i** option.

-k Attempt to skip corrupted file headers and I/O errors that may be encountered. If you want to copy files from a medium that is corrupted or out of sequence, this option lets you read only those files with good headers. (For *cpio* archives that contain other *cpio* archives, if an error is encountered *cpio* may terminate prematurely. *cpio* will find the next good header, which may be one for a smaller archive, and terminate when the smaller archive's trailer is encountered.) Used only with the **-i** option.

-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.

-M *message*
: Define a message to use when switching media. When you use the **-O** or **-I** options and specify a character special device, you can use this option to define the message that is printed when you reach the end of the medium. One **%d** can be placed in the message to print the sequence number of the next medium needed to continue.

-O *file* Direct the output of cpio to *file.* If *file* is a character special device, when the first medium is full replace the medium and type a carriage return to continue to the next medium. Use only with the **-o** option.

-r Interactively *rename* files. If the user types a null line, the file is skipped. If the user types a "." the original pathname will be copied. (Not available with **cpio -p**.)

-s *swap* bytes within each half word. Use only with the **-i** option.

-S *Swap* halfwords within each word. Use only with the **-i** option.

-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)].

-V *Special*Verbose: print a dot for each file seen. Useful to assure the user that **cpio** is working without printing out all file names.

-6 Process an old (i.e., UNIX System *Sixth* Edition format) file. Use only with the **-i** option.

NOTE: *cpio* assumes 4-byte words.

If *cpio* reaches end of medium (end of a diskette for example), when writing to (**-o**) or reading from (**-i**) a character-special device, and **-O** and **-I** aren't used, *cpio* will print the message:

If you want to go on, type device/file name when ready.

To continue, you must replace the medium and type the character special device name (**/dev/rdsk/f0q15dt** for example) and carriage return. You may want to continue by directing *cpio* to use a different device. For example, if you have two floppy drives you may want to switch between them so *cpio* can proceed while you are changing the floppies. (A carriage return alone causes the *cpio* process to exit.)

EXAMPLES

The following examples show three uses of *cpio.*

When standard input is directed through a pipe to **cpio -o**, it groups the files so they can be directed (>) to a single file (**../newfile**). The **c** option insures that the file will be portable to other machines. Instead of *ls*(1), you could use *find*(1), *echo*(1), *cat*(1), etc., to pipe a list of names to *cpio*. You could direct the output to a device instead of a file.

```
ls | cpio -oc >../newfile
```

cpio -i uses the output file of **cpio -o** (directed through a pipe with **cat** in the example), extracts those files that match the patterns (**memo/a1**, **memo/b***), creates directories below the current directory as needed (**-d** option), and places the files in the appropriate directories. The **c** option is used when the file is created with a portable header. If no patterns were given, all files from *newfile* would be placed in the directory.

```
cat newfile | cpio -icd "memo/a1" "memo/b*"
```

cpio -p takes the file names piped to it and copies or links (**-l** option) those files to another directory on your machine (*newdir* in the example). The **-d** options says to create directories as needed. The **-m** option says retain the modification time. [It is important to use the **-depth** option of *find*(1) to generate path names for *cpio*. This eliminates problems *cpio* could have trying to create files under read-only directories.]

```
find . -depth -print | cpio -pdlmv newdir
```

SEE ALSO

cat(1), echo(1), find(1), ls(1), tar(1).
cpio(4) in the *Programmer's Reference Manual.*

NOTES

1) Path names are restricted to 256 characters.
2) Only the super-user can copy special files.
3) Blocks are reported in 512-byte quantities.
4) If a file has 000 permissions, contains more than 0 characters of data, and the user is not root, the file will not be saved or restored.

NAME

crash – examine system images

SYNOPSIS

/etc/crash [**-d** dumpfile] [**-n** namelist] [**-w** outputfile]

DESCRIPTION

The *crash* command is used to examine the system memory image of a live or a crashed system by formatting and printing control structures, tables, and other information. Command line arguments to *crash* are *dumpfile, namelist,* and *outputfile.*

Dumpfile is the file containing the system memory image. The default *dumpfile* is */dev/mem.*

The text file *namelist* contains the symbol table information needed for symbolic access to the system memory image to be examined. The default *namelist* is */unix.* If a system image from another machine is to be examined, the corresponding text file must be copied from that machine.

When the *crash* command is invoked, a session is initiated. The output from a *crash* session is directed to *outputfile.* The default *outputfile* is the standard output.

Input during a *crash* session is of the form:

    function [argument ...]

where *function* is one of the *crash* functions described in the "FUNCTIONS" section of this manual page, and *arguments* are qualifying data that indicate which items of the system image are to be printed.

The default for process-related items is the current process for a running system and the process that was running at the time of the crash for a crashed system. If the contents of a table are being dumped, the default is all active table entries.

The following function options are available to *crash* functions wherever they are semantically valid.

-e Display every entry in a table.

-f Display the full structure.

-p Interpret all address arguments in the command line as *physical* addresses.

-s process
  Specify a process slot other than the default.

-w file Redirect the output of a function to *file.*

Note that if the **-p** option is used, all address and symbol arguments explicitly entered on the command line will be interpreted as physical addresses. If they are not physical addresses, results will be inconsistent.

The functions *mode, defproc,* and *redirect* correspond to the function options **-p, -s,** and **-w.** The *mode* function may be used to set the address translation mode to physical or virtual for all subsequently entered functions; *defproc* sets the value of the process slot argument for subsequent functions;

and *redirect* redirects all subsequent output.

Output from *crash* functions may be piped to another program in the following way:

function [argument ...]!shell_command

For example,

mount ! grep rw

will write all mount table entries with an *rw* flag to the standard output. The redirection option (**-w**) cannot be used with this feature.

Depending on the context of the function, numeric arguments will be assumed to be in a specific radix. Counts are assumed to be decimal. Addresses are always hexadecimal. Table slot arguments are always decimal. Table slot arguments larger than the size of the function table will not be interpreted correctly. Use the *findslot* command to translate from an address to a table slot number. Default bases on all arguments may be overridden. The C conventions for designating the bases of numbers are recognized. A number that is usually interpreted as decimal will be interpreted as hexadecimal if it is preceded by *0x* and as octal if it is preceded by *0*. Decimal override is designated by *0d*, and binary by *0b*.

Aliases for functions may be any uniquely identifiable initial substring of the function name. Traditional aliases of one letter, such as *p* for *proc*, remain valid.

Many functions accept different forms of entry for the same argument. Requests for table information will accept a table entry number or a range. A range of slot numbers may be specified in the form *a-b* where *a* and *b* are decimal numbers. An expression consists of two operands and an operator. An operand may be an address, a symbol, or a number; the operator may be +, −, *, /, &, or ¦ . An operand which is a number should be preceded by a radix prefix if it is not a decimal number (*0* for octal, *0x* for hexidecimal, *0b* for binary). The expression must be enclosed in parentheses ( ). Other functions will accept any of these argument forms that are meaningful.

Two abbreviated arguments to *crash* functions are used throughout. Both accept data entered in several forms. They may be expanded into the following:

table_entry = table entry¦ range

start_addr = address¦ symbol¦ expression

FUNCTIONS

? [**-w** file] List available functions.

!cmd Escape to the shell to execute a command.

adv [**-e**] [**-w** file] [[**-p**]table_entry ...]
Print the advertise table.

base [**-w** file] number ...
Print *number* in binary, octal, decimal, and hexadecimal. A number in a radix other then decimal should be preceded by a prefix that

indicates its radix as follows: *0x,* hexidecimal; *0,* octal; and *0b,* binary.

buffer [-**w** file] [-format] bufferslot

or

buffer [-**w** file] [-format] [-**p**]start_addr
Alias: **b**.
Print the contents of a buffer in the designated format. The following format designations are recognized: **-b**, byte: **-c**, character; **-d**, decimal; **-x**, hexadecimal; **-o**, octal; **-r**, directory; and **-i**, inode. If no format is given, the previous format is used. The default format at the beginning of a *crash* session is hexadecimal.

bufhdr [-**f**] [-**w** file] [[-**p**]table_entry ...]
Alias: **buf**.
Print system buffer headers.

callout [-**w** file]
Alias: **c**.
Print the callout table.

dballoc [-**w** file] [class ...]
Print the dballoc table. If a class is entered, only data block allocation information for that class will be printed.

dbfree [-**w** file] [class ...]
Print free streams data block headers. If a class is entered, only data block headers for the class specified will be printed.

dblock [-**e**] [-**w** file] [-**c** class ...]

or

dblock [-**e**] [-**w** file] [[-**p**] table_entry ...]
Print allocated streams data block headers. If the class option (**-c**) is used, only data block headers for the class specified will be printed.

defproc [-**w** file] [-**c**]

or

defproc [-**w** file] [slot]
Set the value of the process slot argument. The process slot argument may be set to the current slot number (**-c**) or the slot number may be specified. If no argument is entered, the value of the previously set slot number is printed. At the start of a *crash* session, the process slot is set to the current process.

dis [-**w** file] [-**a**] start_addr [count]
Disassemble from the start address for *count* instructions. The default count is 1. The absolute option (**-a**) specifies a non-symbolic disassembly.

ds [-**w** file] virtual_address ...
Print the data symbol whose address is closest to, but not greater than, the address entered.

file [-**e**] [-**w** file] [[-**p**] table_entry ...]
Alias: **f**.
Print the file table.

findaddr [-**w** file] table slot
Print the address of *slot* in *table*. Only tables available to the *size* function are available to *findaddr*.

findslot [-**w** file] virtual_address ...
Print the table, entry slot number, and offset for the address entered. Only tables available to the *size* function are available to *findslot*.

fs [-**w** file] [[-**p**] table_entry ...]
Print the file system information table.

gdp [-**e**] [-**f**] [-**w** file] [[-**p**] table_entry ...]
Print the gift descriptor protocol table.

gdt [-**e**] [-**w** file] [[-**p**] table_entry ...]
Print the global descriptor table.

help [-**w** file] function ...
Print a description of the named function, including syntax and aliases.

idt [-**e**] [-**w** file] [[-**p**] table_entry ...]
Print the interrupt descriptor table.

inode [-**e**] [-**f**] [-**w** file] [[-**p**] table_entry ...]
Alias: **i**.
Print the inode table, including file system switch information.

kfp [-**w** file] [value]
Print the frame pointer for the start of a kernel stack trace. If the value argument is supplied, the kfp is set to that value.

lck [-**e**] [-**w** file] [[-**p**] table_entry ...]
Alias: **l**.
Print record-locking information. If the **-e** option is used or table address arguments are given, the record lock list is printed. If no argument is entered, information on locks relative to inodes is printed.

ldt [-**e**] [-**w** file] [-**s** process] [[-**p**] table_entry ...]
Print the local descriptor table for the given process, or for the current process if none is given.

linkblk [-**e**] [-**w** file] [[-**p**] table_entry ...]
Print the linkblk table.

map [-**w** file] mapname ...
Print the map structure of the given mapname.

mbfree [-**w** file]
Print free streams message block headers.

mblock [**-e**] [**-w** filename] [[**-p**] table_entry ...]
Print allocated streams message block headers.

mode [**-w** file] [mode]
Set address translation of arguments to virtual (**v**) or physical (**p**) mode. If no mode argument is given, the current mode is printed. At the start of a *crash* session, the mode is virtual.

mount [**-e**] [**-w** file] [[**-p**] table_entry ...]
Alias: **m**.
Print the mount table.

nm [**-w** file] symbol ...
Print value and type for the given symbol.

od [**-p**] [**-w** file] [-format] [-mode] [**-s** process] start_addr [count]
Alias: **rd**.
Print *count* values starting at the start address in one of the following formats: character (**-c**), decimal (**-d**), hexadecimal (**-x**), octal (**-o**), ASCII (**-a**), or hexadecimal/character (**-h**), and one of the following modes: long (**-l**), short (**-t**), or byte (**-b**). The default mode for character and ASCII formats is byte; the default mode for decimal, hexadecimal, and octal formats is long. The format **-h** prints both hexadecimal and character representations of the addresses dumped; no mode needs to be specified. When format or mode is omitted, the previous value is used. At the start of a *crash* session, the format is hexadecimal and the mode is long. If no count is entered, 1 is assumed.

panic
Print the latest system notices, warnings, and panic messages from the limited circular buffer kept in memory.

pcb [**-w** file] [process]
Print the process control block (TSS) for the given process. If no arguments are given, the active TSS for the current process is printed.

pdt [**-e**] [**-w** file] [**-s** process] [**-p**] start_addr [count]
The page descriptor table of the designated memory *section* and *segment* is printed. Alternatively, the page descriptor table starting at the start address for *count* entries is printed. If no count is entered, 1 is assumed.

pfdat [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print the pfdata table.

proc [-e] [**-f**] [**-w** file] [[**-p**] table_entry ... **#**procid ...]

or

proc [-**f**] [**-w** file] [**-r**]
Alias: **p**.
Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and process ids may be entered. Each process id must be preceded by a **#**. Alternatively,

process table information for runnable processes may be specified with the runnable option (**-r**). The full option (**-f**) details most of the information in the process table as well as the region table for that process.

qrun [**-w** file]
Print the list of scheduled streams queues.

queue [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print streams queues.

quit Alias: **q**.
Terminate the *crash* session.

rcvd [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the receive descriptor table.

redirect [**-w** file] [**-c**]

or

redirect [**-w** file] [file]
Used with a file name, redirects output of a *crash* session to the named file. If no argument is given, the file name to which output is being redirected is printed. Alternatively, the close option (**-c**) closes the previously set file and redirects output to the standard output.

region [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print the region table.

sdt [**-e**] [**-w** file] [**-s** process] section

or

sdt [**-e**] [**-w** file] [**-s** process] [**-p**] start_addr [count]
The segment descriptor table for the current process is printed.

search [**-p**] [**-w** file] [**-m** mask] [**-s** process] pattern start_addr count
Print the long words in memory that match *pattern*, beginning at the start address for *count* long words. The mask is anded (&) with each memory word and the result compared against the pattern. The mask defaults to 0xffffffff.

size [**-w** file] [**-x**] [structure_name ...]
Print the size of the designated structure. The (**-x**) option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

sndd [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the send descriptor table.

srmount [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print the server mount table.

stack [**-w** file] [process]
Alias: **s**.
Dump stack. If no arguments are entered, the kernel stack for the current process is printed. The interrupt stack and the stack for the

current process are not available on a running system.

stat [**-w** file]
Print system statistics.

stream [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the streams table.

strstat [**-w** file]
Print streams statistics.

trace [**-w** file] [**-r**] [process]
Alias: **t**.
Print kernel stack trace. The kfp value is used with the **-r** option.

ts [**-w** file] virtual_address ...
Print closest text symbol to the designated address.

tty [**-e**] [**-f**] [**-w** file] [**-t** type [[**-p**] table_entry ...]]
Valid types: **co**, **c1**, **c2** (console, com1, com2).
Print the tty table. If no arguments are given, the tty table for the console is printed. If the **-t** option is used, the table for the single tty type specified is printed. If no argument follows the type option, all entries in the table are printed. A single tty entry may be specified from the start address.

user [**-f**] [**-w** file] [process]
Alias: **u**.
Print the ublock for the designated process.

var [**-w** file]
Alias: **v**.
Print the tunable system parameters.

vtop [**-w** file] [**-s** process] start_addr ...
Print the physical address translation of the virtual start address.

FILES

/dev/mem	system image of currently running system

NAME

cron – clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

The *cron* command executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in *crontab* files in the directory **/usr/spool/cron/crontabs**. Users can submit their own *crontab* file via the *crontab*(1) command. Commands which are to be executed only once may be submitted via the *at*(1) command.

The *cron* command only examines *crontab* files and *at* command files during process initialization and when a file changes via *crontab* or *at*. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Since *cron* never exits, it should be executed only once. This is done routinely through **/etc/rc2.d/S75cron** at system boot time. **/usr/lib/cron/FIFO** is used as a lock file to prevent the execution of more than one *cron*.

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/FIFO	used as a lock file
/usr/lib/cron/log	accounting information
/usr/spool/cron	spool area

SEE ALSO

at(1), crontab(1), sh(1).

DIAGNOSTICS

A history of all actions taken by *cron* are recorded in **/usr/lib/cron/log.**

NAME

crontab – user crontab file

SYNOPSIS

crontab [file]
crontab -r
crontab -l

DESCRIPTION

The *crontab* command copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The –r option removes a user's crontab from the crontab directory. *crontab* –l will list the crontab file for the invoking user.

Users are permitted to use *crontab* if their names appear in the file **/usr/lib/cron/cron.allow.** If that file does not exist, the file **/usr/lib/cron/cron.deny** is checked to determine if the user should be denied access to *crontab.* If neither file exists, only root is allowed to submit a job. If **cron.allow** does not exist and **cron.deny** exists but is empty, global usage is permitted. The allow/deny files consist of one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (0–59),
hour (0–23),
day of the month (1–31),
month of the year (1–12),
day of the week (0–6 with 0=Sunday).

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your **$HOME** directory with an **arg0** of **sh.** Users who desire to have their *.profile* executed must explicitly do so in the crontab file. *Cron* supplies a default environment for every shell, defining **HOME, LOGNAME, SHELL(=/bin/sh),** and **PATH(=:/bin:/usr/bin:/usr/lbin).**

If you do not redirect the standard output and standard error of your commands, any generated output or errors will be mailed to you.

FILES

/usr/lib/cron	main cron directory
/usr/spool/cron/crontabs	spool area
/usr/lib/cron/log	accounting information
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users

SEE ALSO

cron(1M), sh(1).

WARNINGS

If you inadvertently enter the **crontab** command with no argument(s), do not attempt to get out with a CTRL-d. This will cause all entries in your **crontab** file to be removed. Instead, exit with a DEL.

NAME

crypt – encode/decode

SYNOPSIS

crypt [password]
crypt [**-k**]

DESCRIPTION

The *crypt* command reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no argument is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. If the **-k** option is used, *crypt* will use the key assigned to the environment variable CRYPTKEY. The *crypt* command encrypts and decrypts with the same key:

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

Files encrypted by *crypt* are compatible with those treated by the editors *ed*(1), *edit*(1), *ex*(1), and *vi*(1) in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; "sneak paths" by which keys or clear text can become visible must be minimized.

The *crypt* command 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.

If the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. The choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES

/dev/tty	for typed key

SEE ALSO

ed(1), edit(1), ex(1), makekey(1), ps(1), stty(1), vi(1).

WARNING

This command is provided with the Security Administration Utilities, which is only available in the United States. 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

The *csplit* command reads *file* and separates it into n+1 sections, defined by the arguments *arg1*. . . *argn*. By default the sections are placed in xx00 . . . xx*n* (*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*.

If the *file* argument is a –, then standard input is used.

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 *prefix***00** . . . *prefixn*. The default is **xx00** . . . **xx***n*.

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

/rexp/	A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional + or – some number of lines (e.g., **/Page/-5**).
%rexp%	This argument is the same as */rexp/*, 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 *rexp* 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 *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *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) in the *Programmer's Reference Manual.*

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 [**-wn**] [**-xn**] [**-h**] [**-v**] [**-s**speed] telno ...

DESCRIPTION

The *ct* command dials the telephone 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. (The set of legal characters for *telno* is 0 through 9, –, =, *, and #. The maximum length *telno* is 31 characters). If more than one telephone number is specified, The *ct* command 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/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 **-w***n* option, where *n* is the maximum number of minutes that *ct* is to wait for a line.

The **-x***n* option is used for debugging; it produces a detailed output of the program execution on stderr. The debugging level, *n*, is a single digit; **-x9** is the most useful value.

Normally, *ct* will hang up the current line, so the line can answer the incoming call. The **-h** option will prevent this action. The **-h** option will also wait for the termination of the specified *ct* process before returning control to the user's terminal. 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 1200.

After the user on the destination terminal logs out, there are two things that could occur depending on what type of getty is on the line (*getty* or *uugetty*). For the first case, *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. In the second case, there is already a getty (*uugetty*) on the line, so the **login:** message will appear.

To log out properly, the user must type **control D**.

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

FILES

/usr/lib/uucp/Devices
/usr/adm/ctlog

SEE ALSO

cu(1C), getty(1M), login(1), uucp(1C), uugetty(1M).

BUGS

For a shared port, one used for both dial-in and dial-out, the *uugetty* program running on the line must have the **-r** option specified [see *uugetty*(1M)].

NAME

cu – call another UNIX system

SYNOPSIS

cu [-sspeed] [-lline] [-**h**] [-**t**] [-**d**] [-**o** ¦ -**e**] [-**n**] telno
cu [-**s** speed] [-**h**] [-**d**] [-**o** ¦ -**e**] -**l** line
cu [-**h**] [-**d**] [-**o** ¦ -**e**] systemname

DESCRIPTION

The *cu* command calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of ASCII files.

The *cu* command accepts the following options and arguments:

-s*speed* Specifies the transmission speed (300, 1200, 2400, 4800, 9600); The default value is "Any" speed which will depend on the order of the lines in the **/usr/lib/uucp/Devices** file. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

-l*line* Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line having the right speed. When the **-l** option is used without the **-s** option, the speed of a line is taken from the Devices file. When the **-l** and **-s** options are both used together, cu will search the Devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made. The specified device is generally a directly connected asynchronous line (e.g., **/dev/tty***ab*) in which case a telephone number (*telno*) is not required. The specified device need not be in the **/dev** directory. If the specified device is associated with an auto dialer, a telephone number must be provided. Use of this option with *systemname* rather than *telno* will not give the desired result (see *systemname* below).

-h Emulates local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.

-t Used to dial an ASCII terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.

-d Causes diagnostic traces to be printed.

-o Designates that odd parity is to be generated for data sent to the remote system.

-n For added security, will prompt the user to provide the telephone number to be dialed rather than taking it from the command line.

-e Designates that even parity is to be generated for data sent to the remote system.

telno	When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.
systemname	A uucp system name may be used rather than a telephone number; in this case, *cu* will obtain an appropriate direct line or telephone number from **/usr/lib/uucp/Systems.** Note: the *systemname* option should not be used in conjunction with the **-l** and **-s** options as *cu* will connect to the first available line for the system name specified, ignoring the requested line and speed.

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 system so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following user-initiated commands:

~.	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.
~%cd	change the directory on the local system. Note: **~!cd** will cause the command to be run by a sub-shell, probably not what was intended.
~%take *from* [*to*]	copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
~%put *from* [*to*]	copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places. For both **~%take** and **put** commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.
~~ *line*	send the line ~ *line* to the remote system.
~%break	transmit a **BREAK** to the remote system (which can also be specified as **~%b**).
~%debug	toggles the **-d** debugging option on or off (which can also be specified as **~%d**).
~t	prints the values of the termio structure variables for the user's terminal (useful for debugging).

~l	prints the values of the termio structure variables for the remote communication line (useful for debugging).
~%nostop	toggles between DC3/DC1 input control protocol and no input control. 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. Internally the program accomplishes this by initiating an output diversion to a file when a line from the remote begins with ~.

Data from the remote is diverted (or appended, if >> is used) to *file* on the local system. The trailing ~> marks the end of the diversion.

The use of **~%put** requires *stty*(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 these current control characters 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, *tabs* mode [see *stty(1)]* should be set on the remote system if tabs are to be copied without expansion to spaces.

When *cu* is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using ~~. Executing a tilde command reminds the user of the local system uname. For example, uname can be executed on Z, X, and Y as follows:

```
uname
Z
~[X]!uname
X
~~[Y]!uname
Y
```

In general, ~ causes the command to be executed on the original machine, ~~ causes the command to be executed on the next machine in the chain.

EXAMPLES

To dial a system whose telephone number is 9 201 555 1212 using 1200 baud (where dialtone is expected after the 9):

```
cu -s1200 9=12015551212
```

If the speed is not specified, "Any" is the default value.

To log in to a system connected by a direct line, enter:

```
cu -l /dev/ttyXX
```

or

```
cu -l ttyXX
```

To dial a system with the specific line and a specific speed, enter:

```
cu -s1200 -l ttyXX
```

To dial a system using a specific line associated with an auto dialer, enter:

```
cu -l culXX 9=12015551212
```

To use a system name, enter:

```
cu systemname
```

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Devices
/usr/spool/locks/LCK..(tty-device)

SEE ALSO

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

DIAGNOSTICS

Exit code is zero for normal exit, otherwise, one.

WARNINGS

The *cu* command does not do any integrity checking on data it transfers. Data fields with special *cu* characters may not be transmitted properly. Depending on the interconnection hardware, it may be necessary to use a **~.** to terminate the conversion even if **stty 0** has been used. Non-printing characters are not dependably transmitted using either the **~%put** or **~%take** commands. *cu* between some modems will not return a login prompt immediately upon connection. A carriage return will return the prompt.

BUGS

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 **-c**list [file ...]
cut **-f**list [**-d**char] [**-s**] [file ...]

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. In addition, a file name of "–" explicitly refers to standard input.

The meanings of the options are:

list	A comma-separated list of integer field numbers (in increasing order), with optional - to indicate ranges [e.g., **1,4,7**; **1-3,8**; **-5,10** (short for **1-5,10**); or **3-** (short for third through last field)].
-c*list*	The *list* following **-c** (no space) specifies character positions (e.g., **-c1-72** would pass the first 72 characters of each line).
-f*list*	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.
-d*char*	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.

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

ERROR: line too long A line can have no more than 1023 characters or fields, or there is no new-line character.

ERROR: bad list for c / f option
Missing **-c** or **-f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

ERROR: no fields	The *list* is empty.
ERROR: no delimeter	Missing *char* on **-d** option.
ERROR: cannot handle multiple adjacent backspaces	Adjacent backspaces cannot be processed correctly.
WARNING: cannot open <filename>	Either *filename* cannot be read or does not exist. If multiple file names are present, processing continues.

SEE ALSO

grep(1), paste(1).

NAME

date – print and set the date

SYNOPSIS

date [+format]
date [mmddhhmm[[yy] ¦ [ccyy]]]

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 (only by super-user). 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; *cc* is the century minus one and is optional; *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 saving time. Only the super-user may change the date.

If the argument begins with +, the output of *date* is under the control of the user. 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. If the argument contains embedded blanks it must be quoted (see the **EXAMPLE** section).

Specifications of native language translations of month and weekday names are supported. The language used depends on the value of the environment variable **LANGUAGE** [see *environ*(5)]. The month and weekday names used for a language are taken from strings in the file for that language in the **/lib/cftime** directory [see *cftime*(4)].

After successfully setting the date and time, *date* will display the new date according to the format defined in the environment variable **CFTIME** [see *environ*(5)].

Field Descriptors (must be preceded by a %):

a	abbreviated weekday name
A	full weekday name
b	abbreviated month name
B	full month name
d	day of month – 01 to 31
D	date as mm/dd/yy
e	day of month – 1 to 31 (single digits are preceded by a blank)
h	abbreviated month name (alias for %b)
H	hour – 00 to 23
I	hour – 01 to 12
j	day of year – 001 to 366
m	month of year – 01 to 12

M	minute – 00 to 59
n	insert a new-line character
p	string containing ante-meridiem or post-meridiem indicator (by default, AM or PM)
r	time as *hh:mm:ss pp* where *pp* is the ante-meridiem or post-meridiem indicator (by default, AM or PM)
R	time as hh:mm
S	second – 00 to 59
t	insert a tab character
T	time as *hh:mm:ss*
U	week number of year (Sunday as the first day of the week) – 01 to 52
w	day of week – Sunday = 0
W	week number of year (Monday as the first day of the week) – 01 to 52
x	Country-specific date format
X	Country-specific time format
y	year within century – 00 to 99
Y	year as *ccyy* (4 digits)
Z	timezone name

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 are not 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

NOTE

Administrators should note the following: if you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending), and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

SEE ALSO

cftime(4), environ(5).

NAME

dc – desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

The *dc* command 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. [see *bc*(1), a preprocessor for *dc* that provides infix notation and a C-like syntax that implements functions. *Bc* also provides reasonable control structures for programs.] 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.

s*x* 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.

l*x* 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.

Q Exits the program. 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 system 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*(1) for array operations.

EXAMPLE

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1).

DIAGNOSTICS

x is unimplemented
: where *x* is an octal number.

stack empty
: for not enough elements on the stack to do what was asked.

Out of space
: when the free list is exhausted (too many digits).

Out of headers
: for too many numbers being kept around.

Out of pushdown
: for too many items on the stack.

Nesting Depth
: for too many levels of nested execution.

NAME

dcopy – copy file systems for optimal access time

SYNOPSIS

/etc/dcopy [-**s**X] [-**a**n] [-**d**] [-**v**] [-**f**fsize[:isize]] inputfs outputfs

DESCRIPTION

The *dcopy* command copies file system *inputfs* to *outputfs*. *Inputfs* is the device file for the existing file system; *outputfs* is the device file to hold the reorganized result. For the most effective optimization, *inputfs* should be the raw device and *outputfs* should be the block device. Both *inputfs* and *outputfs* should be unmounted file systems.

With no options, *dcopy* copies files from *inputfs* compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap. The possible options are

-s*X*	supply device information for creating an optimal organization of blocks in a file. The forms of *X* are the same as the **-s** option of *fsck*(1M).
-a*n*	place the files not accessed in *n* days after the free blocks of the destination file system (default for *n* is 7). If no *n* is specified, then no movement occurs.
-d	leave order of directory entries as is (default is to move subdirectories to the beginning of directories).
-v	currently reports how many files were processed, and how big the source and destination freelists are.
-f*fsize*[:*isize*]	specify the *outputfs* file system and inode list sizes (in blocks). If the option (or *:isize*) is not given, the values from the *inputfs* are used.

dcopy catches interrupts and quits, and reports on its progress. To terminate *dcopy* send a quit signal, followed by an interrupt or quit.

SEE ALSO

fsck(1M), mkfs(1M), ps(1).

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

The *dd* command 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.

option	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs;* also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input blocks before starting copy
seek=*n*	seek *n* blocks from beginning of output file before copying
count=*n*	copy only *n* input blocks
conv=**ascii**	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input block to *ibs*
..., ...	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 multiplication.

The *cbs* is used only if *conv=ascii* or *conv=ebcdic* is specified. In the former case, *cbs* characters are placed into the conversion buffer (converted to ASCII). Trailing blanks are trimmed and a 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). Blanks are added to make up an output block of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

DIAGNOSTICS

f+p blocks in(out)	numbers of full and partial blocks read(written)

NAME

deroff – remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [**-m**x] [**-w**] [files]

DESCRIPTION

The *deroff* command 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 **.nx** 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 to 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."

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

devnm – device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

The *devnm* command identifies the special file associated with the mounted file system where the argument *name* resides.

This command is most commonly used by **/etc/brc** [see *brc*(1M)] to construct a mount table entry for the **root** device.

EXAMPLE

The command:

```
        /etc/devnm /usr
```

produces

```
        /dev/dsk/0s3
```

if **/usr** is mounted on **/dev/dsk/0s3**.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M).

NAME

df – report number of free disk blocks and i-nodes

SYNOPSIS

df [**-lt**] [**-f**] [*file-system* ¦ *directory* ¦ *mounted-resource*]

DESCRIPTION

The **df** command prints out the number of free blocks and free i-nodes in mounted file systems, directories, or mounted resources by examining the counts kept in the super-blocks.

The *file-system* may be specified either by device name (e.g., **/dev/dsk/0s1**) or by mount point directory name (e.g., **/usr**).

directory can be a directory name. The report presents information for the device that contains the directory.

mounted-resource can be a remote resource name. The report presents information for the remote device that contains the resource.

If no arguments are used, the free space on all locally and remotely mounted file systems is printed.

The **df** command uses the following options:

-l only reports on local file systems.

-t causes the figures for total allocated blocks and i-nodes to be reported as well as the free blocks and i-nodes.

-f an actual count of the blocks in the free list is made, rather than taking the figure from the super-block (free i-nodes are not reported). This option will not print any information about mounted remote resources.

NOTE

If multiple remote resources are listed that reside on the same file system on a remote machine, each listing after the first one will be marked with an asterisk.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

mount(1M).
fs(4), mnttab(4) in the *Programmer's Reference Manual.*

NAME

diff – differential file comparator

SYNOPSIS

diff [**-efbh**] file1 file2

DESCRIPTION

The *diff* command 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

bdiff(1), 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 (.).

WARNINGS

Missing newline at end of file X
indicates that the last line of file X did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [**-ex3**] file1 file2 file3

DESCRIPTION

The *diff3* command 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 will not work.

NAME

dircmp – directory comparison

SYNOPSIS

dircmp [**-d**] [**-s**] [**-w***n*] dir1 dir2

DESCRIPTION

The *dircmp* command 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 file names 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.

-w*n* Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

cmp(1), diff(1).

NAME

diskadd – disk partitioning utility

SYNOPSIS

/etc/diskadd [disk_number]

DESCRIPTION

This shell script (in conjunction with the program **/etc/adddisk**) allows the system administrator to set up additional disks for use by the UNIX system. (The initial system disk is configured during system installation.) It is an interactive program which prompts the user for information about the setup of the disk and allows specification of known media defects. It allows the partitioning of the disk into a *tmp* filesystem (if desired), additional swap/paging space (if desired), and from 1 to 4 user filesystems. All input is double-checked with the user for correctness. The user is then asked whether the filesystems on the new disk should be mounted automatically. Finally, the disk is verified and filesystems are created. If automatic mounting was specified, directories are created in the root filesystem to hold the new filesystems, they are mounted, and **/etc/fstab** is updated to re-mount them on subsequent bootups of the system. Device and partition stanzas for the new disk are appended to the **/etc/partitions** file for use with *mkpart*(1M).

Prior to running *diskadd* the system administrator must create special files in **/dev/dsk** and **/dev/rdsk** for accessing the new disk. The names of these files are of the form **/dev/***[r]***dsk/***[cid]***jsk**, where 'i' is the controller number ('cid' is omitted for controller 0), 'j' is the drive number on the controller, and 'k' is the partition number on the disk. The minimum set of special files for a new disk is a character (rdsk) device for partition 0 (**/dev/rdsk/1s0** for the second drive on the first controller), and block (dsk) device for each partition added. Partition 1 is always *tmp* (if specified), partition 2 is always *swap* (if specified), and new user filesystems are made in partitions 3-6.

The *fdisk* program will be run by *diskadd* to create a DOS-style partition table. The user is expected to use *fdisk* to create a UNIX partition. By default, *diskadd* will set up **/dev/rdsk/1s0** (the second disk on Controller 0). A different disk can be used by giving one of the following arguments to *diskadd*:

/dev/rdsk/1s0 (default)
/dev/rdsk/c1d0s0
/dev/rdsk/c1d1s0

During the execution of the *mkpart* command, warnings about having no root partition from which to boot will be issued. These may be ignored, as added disks are not bootable.

If swap space is added on the new drive, it must be made available for system use with the *swap*(1M) program. Diskadd doesn't do this, as there is no automatic facility for adding swap space on bootup.

FILES

/tmp/partitions
/tmp/addparts
/tmp/mkfs.data
/tmp/diskname
/etc/partitions
/dev/rdsk/*s0
/dev/dsk/*s?
/etc/fstab

SEE ALSO

mkfs(1M), mknod(1), mkpart(1M), swap(1M).

NAME

diskusg – generate disk accounting data by user ID

SYNOPSIS

diskusg [options] [files]

DESCRIPTION

diskusg generates intermediate disk accounting information from data in *files*, or the standard input if omitted. *diskusg* output lines on the standard output, one per user, in the following format: *uid login #blocks*

where

uid -	the numerical user ID of the user.
login -	the login name of the user; and
#blocks -	the total number of disk blocks allocated to this user.

diskusg normally reads only the i-nodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices.

diskusg recognizes the following options:

-s the input data is already in *diskusg* output format. *diskusg* combines all lines for a single user into a single line.

-v verbose. Print a list on standard error of all files that are charged to no one.

-i *fnmlist* ignore the data on those file systems whose file system name is in *fnmlist*. *Fnmlist* is a list of file system names separated by commas or enclosed within quotes. *diskusg* compares each name in this list with the file system name stored in the volume ID [see *labelit*(1M)].

-p *file* use *file* as the name of the password file to generate login names. **/etc/passwd** is used by default.

-u *file* write records to *file* of files that are charged to no one. Records consist of the special file name, the i-node number, and the user ID.

The output of *diskusg* is normally the input to *acctdisk* [see *acct*(1M)] which generates total accounting records that can be merged with other accounting records. *diskusg* is normally run in *dodisk* [see *acctsh*(1M)].

EXAMPLES

The following will generate daily disk accounting information:

```
for i in /dev/dsk/0s1 /dev/dsk/0s3; do
        diskusg $i > dtmp.'basename $i' &
done
wait
diskusg -s dtmp.* | sort +0n +1 | acctdisk > disktacct
```

FILES

/etc/passwd used for user ID to login name conversions

SEE ALSO

acct(1M), acctsh(1M)
acct(4) in the *Programmer's Reference Manual*

NAME

displaypkg – display installed packages

SYNOPSIS

displaypkg

DESCRIPTION

The *displaypkg* command will list the names of all the packages that were installed using the *installpkg* command.

SEE ALSO

installpkg(1), removepkg(1).

NAME

dname – Print Remote File Sharing domain and network names

SYNOPSIS

dname [–D *domain*] [–N *netspec*] [–dna]

DESCRIPTION

The **dname** command prints or defines a host's Remote File Sharing domain name or the network used by Remote File Sharing as transport provider. When used with **d**, **n**, or **a** options, **dname** can be run by any user to print the domain name, network name, or both, respectively. Only a user with root permission can use the **-D** *domain* option to set the domain name for the host or **-N** *netspec* to set the network specification used for Remote File Sharing. (The value of *netspec* is the network device name, relative to the */dev* directory. For example, the STARLAN NETWORK uses **starlan.**)

The *domain* must consist of no more than 14 characters, consisting of any combination of letters (upper and lower case), digits, hyphens (-), and underscores (_)

When **dname** is used to change a domain name, the host's password is removed. The administrator will be prompted for a new password the next time Remote File Sharing is started [*rfstart*(1M)].

If **dname** is used with no options, it will default to **dname -d**.

ERRORS

You cannot use the **-N** or **-D** options while Remote File Sharing is running.

SEE ALSO

rfstart(1M).

NAME

du – summarize disk usage

SYNOPSIS

du [**-sar**] [names]

DESCRIPTION

The *du* command reports 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, the current directory is used.

The optional arguments are as follows:

-s causes only the grand total (for each of the specified *names*) to be given.

-a causes an output line to be generated for each file.

If neither **-s** or **-a** is specified, an output line is generated for each directory only.

-r will cause *du* to generate messages about directories that cannot be read, files that cannot be opened, etc., rather than being silent (the default).

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 links between files in different directories where the directories are on separate branches of the file system hierarchy, *du* will count the excess files more than once.

Files with holes in them will get an incorrect block count. (See Chapter 5, File System Administration, in the *System Administrator's Guide*.)

NAME

echo – echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

The *echo* command 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 \:

b	backspace
c	print line without new-line
f	form-feed
n	new-line
r	carriage return
t	tab
v	vertical tab
\\\\	backslash
0*n*	where *n* is the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number representing that character.

The *echo* command is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

CAVEATS

When representing an 8-bit character by using the escape convention \**0***n*, the *n* must **always** be preceded by the digit zero (0).

For example, typing: **echo 'WARNING:\07'** will print the phrase **WARNING:** and sound the "bell" on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the "\" that precedes the "07".

For the octal equivalents of each character, see ascii(5) in the *Programmer's Reference Manual.*

NAME

ed, red – text editor

SYNOPSIS

ed [–s] [–**p** string] [–x] [–C] [file]

red [–s] [–**p** string] [–x] [–C] [file]

DESCRIPTION

ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see the following text) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited.

–s 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*.

–p Allows the user to specify a prompt string.

–x Encryption option; when used, *ed* simulates an **X** command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of *crypt*(1). The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **–x** option. See *crypt*(1). Also, see the **WARNINGS** section at the end of this manual page.

–C Encryption option; the same as the **–x** option, except that *ed* simulates a **C** command. The **C** command is like the **X** command, except that all text read in is assumed to have been encrypted.

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 allow editing of files only 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: When you are entering text into the file, this format is not in effect; instead, because of being in **stty –tabs** or **stty tab3** mode, tabs are expanded to every eighth column.

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. Leave input mode by typing a period (.) at the beginning of a line, followed immediately by a carriage return.

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. **$** (dollar sign), 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 *RE*s 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 dollar sign (**$**) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction **^***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 an ASCII lower-case letter (**a**-**z**). 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**.

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**.

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 *a*ppend 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 new-line character).

(.)c
<text>
.

The *c*hange 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.

C

Same as the **X** command, except that *ed* assumes all text read in for the **e** and **r** commands is encrypted unless a null key is typed in.

(.,.)d

The *d*elete 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 *e*dit 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.**

E *file*

The *E*dit 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 *f*ile-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 *g*lobal 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.**

(1,$)G/*RE***/**
In the interactive *G*lobal 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 *h*elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H
The *H*elp 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 *i*nsert 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 new-line character).

(.,.+1)j
The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*
The mar*k* command marks the addressed line with name *x*, which must be an ASCII lower-case letter (**a-z**). The address '*x* then

addresses this line; . is unchanged.

(.,.)l
The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab, backspace*) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any command other than *e, f, r,* or *w*.

(.,.)m*a*
The *m*ove 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 *n*umber 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 command other than *e, f, r,* or *w*.

(.,.)p
The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any command other than *e, f, r,* or *w*. For example, *dp* deletes the current line and prints the new current line.

P
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 *q*uit command causes *ed* to exit. No automatic write of a file is done; however, see **DIAGNOSTICS.**

Q
The editor exits without checking if changes have been made in the buffer since the last *w* command.

($)r *file*
The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently remembered file name, if any, is used (see *e* and *f* commands). The currently remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell [*sh*(1)] command whose output is to be read. For example, "$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** or
(.,.)s/*RE***/***replacement***/n** n = 1–512

The *s*ubstitute 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. If a number n appears after the command, only the n-th occurrence of the matched string on each addressed line 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**.

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.

(.,.)t*a*

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 *u*ndo 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 *w*rite 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 *umask*(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 is prompted for, and it is used in subsequent **e**, **r**, and **w** commands to decrypt and encrypt text using the *crypt*(1) algorithm. An educated guess is made to determine whether text read in for the **e** and **r** commands is encrypted. A null key turns off encryption. Subsequent *e*, *r*, and *w* commands will use this key to encrypt or decrypt the text [see *crypt*(1)]. An explicitly empty key turns off encryption. Also, see the **-x** option of *ed*.

($)=
: 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 system 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 in a line, 256 characters in a global command list, and 64 characters in the pathname of a file (counting slashes). 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.

If a file is not terminated by a new-line character, **ed** adds one and puts out a message explaining what it did.

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

$TMPDIR	if this environmental variable is not null, its value is used in place of **/usr/tmp** as the directory name for the temporary work file.
/usr/tmp	if **/usr/tmp** exists, it is used as the directory name for the temporary work file.
/tmp	if the environmental variable **TMPDIR** does not exist or is null, and if **/usr/tmp** does not exist, then **/tmp** is used as the directory name for the temporary work file.
ed.hup	work is saved here if the terminal is hung up.

NOTES

The **-** option, although it continues to be supported, has been replaced in the documentation by the **-s** option that follows the Command Syntax Standard [see *intro*(1)].

SEE ALSO

edit(1), ex(1), grep(1), sed(1), sh(1), stty(1), umask(1), vi(1).
fspec(4), regexp(5) in the *Programmer's Reference Manual.*

DIAGNOSTICS

?	for command errors.
?*file*	for an inaccessible file. (use the *h*elp and *H*elp 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 **-s** command-line option inhibits this feature.

WARNINGS

The encryption options and commands are provided with the Security Administration Utilities package, which is available only in the United States.

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 editor is invoked from a restricted shell [see *sh*(1)].

The sequence **n** in a RE does not match a new-line character.

If the editor input is coming from a command file (e.g., ed file < ed-cmd-file), the editor will exit at the first failure.

NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [**-r**] [**-x**] [**-C**] *name*...

DESCRIPTION

edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor. It operates precisely as *ex*(1) with the following options automatically set:

novice	ON
report	ON
showmode	ON
magic	OFF

These options can be turned on or off via the **set** command in *ex*(1).

-r Recover file after an editor or system crash.

-x Encryption option; when used the file will be encrypted as it is being written and will require an encryption key to be read. *edit* makes an educated guess to determine if a file is encrypted or not. See *crypt*(1). Also, see the **WARNING** section at the end of this manual page.

-C Encryption option; the same as **-x** except that *edit* assumes files are encrypted.

The following brief introduction should help you get started with *edit.* If you are using a CRT terminal you may want to learn about the display editor *vi.*

To edit the contents of an existing file you begin with the command **edit** *name* to the shell. *edit* makes a copy of the file that you can then edit, and tells you how many lines and characters are in the file. To create a new file, you also begin with the command **edit** with a filename: **edit** *name;* the editor will tell you it is a `New File].`

The *edit* command prompt is the colon (**:**), which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit's* buffer (its name for the copy of the file you are editing). When you start editing, *edit* makes the last line of the file the current line. Most commands to *edit* use the current line if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and type carriage return (as you should after all *edit* commands), the current line will be printed. If you **delete** (**d**) the current line, *edit* will print the new current line, which is usually the next line in the file. If you **delete** the last line, then the new last line becomes the current one.

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you execute this command (typing a carriage return after the word **append**), *edit* will read lines from your terminal until you type a line consisting of just a dot (**.**); it places these lines after the current line. The last line you type then becomes the current line.

The command **insert** (**i**) is like **append**, but places the lines you type before, rather than after, the current line.

edit numbers the lines in the buffer, with the first line having number 1. If you execute the command **1**, then *edit* will type the first line of the buffer. If you then execute the command **d**, *edit* will delete the first line, line 2 will become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command: s/*old*/*new*/ where *old* is the string of characters you want to replace and *new* is the string of characters you want to replace *old* with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say `[Modified]` if you have changed the buffer. After modifying a file, you can save the contents of the file by executing a **write** (**w**) command. You can leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will receive the message `No write since last change (:quit! overrides)`, and *edit* will wait for another command. If you do not want to write the buffer out, issue the **quit** command followed by an exclamation point (**q!**). The buffer is then irretrievably discarded and you return to the shell.

By using the **d** and **a** commands and giving line numbers to see lines in the file, you can make any changes you want. You should learn at least a few more things, however, if you will use *edit* more than a few times.

The **change** (**c**) command changes the current line to a sequence of lines you supply (as in **append**, you type lines up to a line consisting of only a dot (**.**). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., **3,5c**. You can print lines this way too: **1,23p** prints the first 23 lines of the file.

The **undo** (**u**) command reverses the effect of the last command you executed that changed the buffer. Thus if you execute a **substitute** command that does not do what you want, type **u** and the old contents of the line will be restored. You can also undo an **undo** command. *edit* will give you a warning message when a command affects more than one line of the buffer. Note that commands such as **write** and **quit** cannot be undone.

To look at the next line in the buffer, type carriage return. To look at a number of lines, type **^D** (while holding down the control key, press **d**) rather than carriage return. This will show you a half-screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at nearby text by executing the **z** command. The current line will appear in the middle of the text displayed, and the last line displayed will become the current line; you can get back to the line where you were before you executed the **z** command by typing ''. The **z** command has other options: **z–** prints a screen of text (or 24 lines) ending where you are; **z+** prints the next screenful. If you want less than a screenful of lines, type **z.11** to display five lines before and five lines after the current line. (Typing **z.***n*, when *n* is an odd number, displays a total of *n* lines, centered about the current line; when *n* is an even number, it displays *n*–1 lines, so that the lines displayed are centered around the current line.) You can give counts after other commands; for example, you can delete 5 lines starting with the current line with the command **d5** .

To find things in the file, you can use line numbers if you happen to know them. Since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backward and forward in the file for strings by giving commands of the form */text/* to search forward for *text* or *?text?* to search backward for *text* . If a search reaches the end of the file without finding *text,* it wraps around and continues to search back to the line where you are. A useful feature here is a search of the form */^text/* which searches for *text* at the beginning of a line. Similarly */text$/* searches for *text* at the end of a line. You can leave off the trailing **/** or **?** in these commands.

The current line has the symbolic name dot (.); this is most useful in a range of lines as in **.,$p** which prints the current line plus the rest of the lines in the file. To move to the last line in the file, you can refer to it by its symbolic name $. Thus the command **$d** deletes the last line in the file, no matter what the current line is. Arithmetic with line references is also possible. Thus the line **$–5** is the fifth before the last and .+**20** is 20 lines after the current line.

You can determine the current line by typing **.=** . This is useful if you wish to move or copy a section of text within a file or between files. Find the first and last line numbers you wish to copy or move. To move lines 10 through 20, type **10,20d a** to delete these lines from the file and place them in a buffer named **a**. *edit* has 26 such buffers named **a** through **z**. To put the contents of buffer **a** after the current line, type **put a**. If you want to move or copy these lines to another file, execute an **edit** (**e**) command after copying the lines; following the **e** command with the name of the other file you wish to edit, i.e., **edit chapter2**. To copy lines without deleting them, use **yank** (**y**) in place of **d**. If the text you wish to move or copy is all within one file, it is not necessary to use named buffers. For example, to move lines 10 through 20 to the end of the file, type **10,20m $**.

SEE ALSO

ed(1), ex(1), vi(1).

WARNING

The encryption options are provided with the Security Administration Utilities package, which is available only in the United States.

NAME

egrep – search a file for a pattern using full regular expressions

SYNOPSIS

egrep [options] full regular expression [file ...]

DESCRIPTION

The *egrep* command (*expression grep*) searches files for a pattern of characters and prints all lines that contain that pattern. *egrep* uses full regular expressions (expressions that have string values that use the full set of alphanumeric and special characters) to match the patterns. It uses a fast deterministic algorithm that sometimes needs exponential space.

The *egrep* command accepts full regular expressions as in *ed*(1), except for **\(** and **\)**, with the addition of:

1. A full regular expression followed by + that matches one or more occurrences of the full regular expression.
2. A full regular expression followed by **?** that matches 0 or 1 occurrences of the full regular expression.
3. Full regular expressions separated by | or by a new-line that match strings that are matched by any of the expressions.
4. A full regular expression that may be enclosed in parentheses **()** for grouping.

Be careful using the characters **$**, *****, **[**, **^**, **|**, **(**, **)**, and \ in *full regular expression,* because they are also meaningful to the shell. It is safest to enclose the entire *full regular expression* in single quotes '...'.

The order of precedence of operators is **[ ]**, then ***?+**, then concatenation, then | and new-line.

If no files are specified, *egrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

-c Print only a count of the lines that contain the pattern.

-i Ignore upper/lower case distinction during comparisons.

-l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.

-n Precede each line by its line number in the file (first line is 1).

-v Print all lines except those that contain the pattern.

-e *special_expression*
Search for a *special expression* (*full regular expression* that begins with a -).

-f *file* Take the list of *full regular expressions* from *file*.

SEE ALSO

ed(1), fgrep(1), grep(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* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h.**

NAME

enable, disable – enable/disable LP printers

SYNOPSIS

enable printers
disable [**-c**] [**-r**[reason]] printers

DESCRIPTION

The *enable* command activates the named *printers,* enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

The *disable* command 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

The *env* command 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) in the *Programmer's Reference Manual.*

NAME

ex – text editor

SYNOPSIS

ex [**-s**] [**-v**] [**-t** tag] [**-r** file] [**-L**] [**-R**] [**-x**] [**-C**] [**-c** command] file ...

DESCRIPTION

ex is the root of a family of editors: *ex* and *vi*. *ex* is a superset of *ed*, with the most notable extension being a display editing facility. Display-based editing is the focus of *vi*.

If you have a CRT terminal, you may wish to use a display-based editor; in this case see *vi*(1), which is a command which focuses on the display-editing portion of *ex*.

For ed Users

If you have used *ed*(1) you will find that, in addition to having all of the *ed*(1) commands available, *ex* has a number of additional features useful on CRT terminals. Intelligent terminals and high-speed terminals are very pleasant to use with *vi*. Generally, the *ex* editor uses far more of the capabilities of terminals than *ed*(1) does and uses the terminal capability data base [see *terminfo*(4)] and the type of the terminal you are using from the environmental variable TERM to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi*(1).

ex contains a number of features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Typing **^D** (control-d) causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just typing return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

ex gives you help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files, unless you edited them, so that you do not accidentally overwrite a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the telephone, you can use the editor **recover** command (or **-r** *file* option) to retrieve your work. This will get you back to within a few lines of where you left off.

ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming file names and is replaced by the name of the current file.

The editor has a group of buffers whose names are the ASCII lower-case letters (**a**-**z**). You can place text in these named buffers where it is available to be inserted elsewhere in the file. The contents of these buffers remain available when you begin editing a new file using the **edit** (**e**) command.

There is a command **&** in *ex* which repeats the last **substitute** command. In addition, there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore the case of letters in searches and substitutions. *ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

ex has a set of options which you can set to tailor it to your liking. One option which is very useful is the **autoindent** option that allows the editor to supply leading white space to align text automatically. You can then use **^D** as a backtab and space or tab to move forward to align new code easily.

Miscellaneous useful features include an intelligent **join** (**j**) command that supplies white space between joined lines automatically, commands "<" and ">" which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort*(1).

Invocation Options

The following invocation options are interpreted by *ex* (previously documented options are discussed in the **NOTES** section at the end of this manual page):

-s	Suppress all interactive-user feedback. This is useful in processing editor scripts.
-v	Invoke *vi*
-t *tag*	Edit the file containing the *tag* and position the editor at its definition.
-r *file*	Edit *file* after an editor or system crash. (Recovers the version of *file* that was in the buffer when the crash occurred.)
-L	List the names of all files saved as the result of an editor or system crash.
-R	**Readonly** mode; the **readonly** flag is set, preventing accidental overwriting of the file.
-x	Encryption option; when used, *ex* simulates an **X** command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of *crypt*(1). The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **-x** option. [See *crypt*(1)]. Also, see the **WARNINGS** section at the end of this manual page.

-C	Encryption option; the same as the **-x** option, except that *ex* simulates a **C** command. The **C** command is like the **X** command, except that all text read in is assumed to have been encrypted.
-c *command*	Begin editing by executing the specified editor *command* (usually a search or positioning command).

The *file* argument indicates one or more files to be edited.

ex States

Command	Normal and initial state. Input prompted for by **:**. Your line kill character cancels a partial command.
Insert	Entered by **a**, **i**, or **c**. Arbitrary text may be entered. Insert state normally is terminated by a line having only **"."** on it, or, abnormally, with an interrupt.
Visual	Entered by typing **vi**; terminated by typing **Q** or ^\ (control-\).

ex Command Names and Abbreviations

abbrev	**ab**	**map**		**set**	**se**
append	**a**	**mark**	**ma**	**shell**	**sh**
args	**ar**	**move**	**m**	**source**	**so**
change	**c**	**next**	**n**	**substitute**	**s**
copy	**co**	**number**	**nu**	**unabbrev**	**unab**
delete	**d**	**preserve**	**pre**	**undo**	**u**
edit	**e**	**print**	**p**	**unmap**	**unm**
file	**f**	**put**	**pu**	**version**	**ve**
global	**g**	**quit**	**q**	**visual**	**vi**
insert	**i**	**read**	**r**	**write**	**w**
join	**j**	**recover**	**rec**	**xit**	**x**
list	**l**	**rewind**	**rew**	**yank**	**ya**

ex Commands

forced encryption	**C**	heuristic encryption	**X**
resubst	**&**	print next	CR
rshift	>	lshift	<
scroll	**^D**	window	**z**
shell escape	**!**		

ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	*?pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
-	previous	*'x*	marked with *x*
+*n*	*n* forward	*''*	previous context
%	1,$		

Initializing options

EXINIT	place **set**'s here in environment variable
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file

set *x*	enable option *x*
set no*x*	disable option *x*
set *x*=*val*	give value *val* to option *x*
set	show changed options
set all	show all options
set *x***?**	show value of option *x*

Most useful options and their abbreviations

autoindent	**ai**	supply indent
autowrite	**aw**	write before changing files
directory	**dir**	specify the directory
exrc	**ex**	allow *vi/ex* to read the **.exrc** in the current directory. This option is set in the **EXINIT** shell variable or in the **.exrc** file in the **$HOME** directory.
ignorecase	**ic**	ignore case of letters in scanning
list		print **^I** for tab, **$** at end
magic		treat **.** **[** ***** special in patterns
modelines		first five lines and last five lines executed as *vi/ex* commands if they are of the form **ex:***command***:** or **vi:***command***:**
number	**nu**	number lines
paragraphs	**para**	macro names that start paragraphs
redraw		simulate smart terminal
report		informs you if the number of lines modified by the last command is greater than the value of the **report** variable
scroll		command mode lines
sections	**sect**	macro names that start sections
shiftwidth	**sw**	for < >, and input **^D**
showmatch	**sm**	to **)** and **}** as typed
showmode	**smd**	show insert mode in *vi*
slowopen	**slow**	stop updates during insert
term		specifies to **vi** the type of terminal being used (the default is the value of the environmental variable **TERM**)
window		visual mode lines
wrapmargin	**wm**	automatic line splitting
wrapscan	**ws**	search around end (or beginning) of buffer

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\\<	beginning of word
\\>	end of word
[*str***]**	any character in *str*
[^*str***]**	any character not in *str*
[*x–y***]**	any character between *x* and *y*
*****	any number of preceding characters

AUTHOR

vi and *ex* are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/exstrings	error messages
/usr/lib/exrecover	recover command
/usr/lib/expreserve	preserve command
/usr/lib/terminfo/*	describes capabilities of terminals
$HOME/.exrc	editor startup file
./.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve/*login*	preservation directory (where *login* is the user's login)

NOTES

Several options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard [see *intro*(1)]. The **-** option has been replaced by **-s**, a **-r** option that is not followed with an option-argument has been replaced by **-L**, and +*command* has been replaced by **-c** *command*.

SEE ALSO

crypt(1), ed(1), edit(1), grep(1), sed(1), sort(1), vi(1).

curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual*.

User's Guide.

"curses/terminfo" chapter of the *Programmer's Guide*.

WARNINGS

The encryption options and commands are provided with the Security Administration Utilities package, which is available only in the United States.

BUGS

The **z** command prints the number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line **-s** option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

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, 2s 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).

DIAGNOSTICS

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.

syntax error	for operator/operand errors
non-numeric argument	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

factor – obtain the prime factors of a number

SYNOPSIS

factor [integer]

DESCRIPTION

When you use *factor* without an argument, it waits for you to give it an integer. After you give it a positive integer less than or equal to 10^{14}, it factors the integer, prints its prime factors the proper number of times, and then waits for another integer. *factor* exits if it encounters a zero or any non-numeric character.

If you invoke *factor* with an argument, it factors the integer as described above, and then it exits.

The maximum time to factor an integer is proportional to $\sqrt{n}$. *factor* will take this time when n is prime or the square of a prime.

DIAGNOSTICS

factor prints the error message, "Ouch," for input out of range or for garbage input.

NAME

fdisk – create or modify hard disk partition table

SYNOPSIS

fdisk

DESCRIPTION

This command is used to create and modify the partition table that is put in the first sector of the hard disk. This table is used by DOS and by the first-stage bootstrap to identify parts of the disk reserved for different operating systems, and to identify the partition containing the second-stage bootstrap (the *active* partition). The optional argument can be used to specify the raw device associated with the hard disk; the default value is **/dev/rdsk/0s0.**

The program displays the partition table as it exists on the disk, and then presents a menu allowing the user to modify the table. The menu, questions, warnings, and error messages are intended to be self-explanatory.

If there is no partition table on the disk, the user is given the option of creating a default partitioning or specifying the initial table values. The default partitioning allows 10% of the disk for MS-DOS and 90% for the UNIX system, and makes the UNIX system partition active. In either case, when the initial table is created, *fdisk* also writes out the first-stage bootstrap code [see *hd*(7)] along with the partition table. After the initial table is created, only the table is changed; the bootstrap is not modified.

Menu Options

The following are the menu options given by the *fdisk* program:

Create a partition
: This option allows the user to create a new partition. The maximum number of partitions is 4. The program will ask for the type of the partition (MS-DOS, UNIX system, or other). It will then ask for the size of the partition as a percentage of the disk. The user may also enter the letter **c** at this point, in which case the program will ask for the starting cylinder number and size of the partition in cylinders. If a **c** is not entered, the program will determine the starting cylinder number where the partition will fit. In either case, if the partition would overlap an existing partition, or will not fit, a message is displayed and the program returns to the original menu.

Change Active (Boot from) partition
: This option allows the user to specify the partition where the first-stage bootstrap will look for the second-stage bootstrap, otherwise known as the *active* partition.

Delete a partition
: This option allows the user to delete a previously created partition. Note that this will destroy all data in that partition.

Exit
: This option writes the new version of the table created during this session with *fdisk* out to the hard disk, and exits the program.

Cancel This option exits without modifying the partition table.

DIAGNOSTICS

Most messages will be self-explanatory. The following may appear immediately after starting the program:

Fdisk: cannot open <device>
: This indicates that the device name argument is not valid.

Fdisk: unable to get device parameters for device <device>
: This indicates a problem with the configuration of the hard disk, or an error in the hard disk driver.

Fdisk: error reading partition table
: This indicates that some error occurred when trying initially to read the hard disk. This could be a problem with the hard disk controller or driver, or with the configuration of the hard disk.

This message may appear after selecting the *Exit* option from the menu.

Fdisk: error writing boot record
: This indicates that some error occurred when trying to write the new partition table out to the hard disk. This could be a problem with the hard disk controller, the disk itself, the driver, or the configuration of the hard disk.

FILES

/dev/rdsk/0s0

SEE ALSO

mkpart(1M), disk(7), hd(7).

WARNING

Compatable with MS-DOS Version 3.2.

NAME

ff – list file names and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

The *ff* command reads the i-list and directories of the *special* file, assuming it is a file system. I-node data is saved for files which match the selection criteria. Output consists of the path name for each saved i-node, plus other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by *ff* is:

```
path-name i-number
```

With all *options* enabled, output fields would be:

```
path-name i-number size uid
```

The argument *n* in the *option* descriptions that follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, -*n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24-hour period.

-I	Do not print the i-node number after each path name.
-l	Generate a supplementary list of all path names for multiply-linked files.
-p *prefix*	The specified *prefix* will be added to each generated path name. The default is **.** (dot).
-s	Print the file size, in bytes, after each path name.
-u	Print the owner's login name after each path name.
-a *n*	Select if the i-node has been accessed in *n* days.
-m *n*	Select if the i-node has been modified in *n* days.
-c *n*	Select if the i-node has been changed in *n* days.
-n *file*	Select if the i-node has been modified more recently than the argument *file*.
-i *i-node-list*	Generate names for only those i-nodes specified in *i-node-list*.

SEE ALSO

find(1), ncheck(1M).

BUGS

If the **-l** option is not specified, only a single path name out of all possible ones is generated for a multiply-linked i-node. If **-l** is specified, all possible names for every linked file on the file system are included in the output. However, no selection criteria apply to the names generated.

NAME

fgrep – search a file for a character string

SYNOPSIS

fgrep [options] string [file ...]

DESCRIPTION

The *fgrep* (fast *grep*) command seaches files for a character string and prints all lines that contain that string. *fgrep* is different from *grep(1)* and *egrep(1)* because it searches for a string, instead of searching for a pattern that matches an expression. It uses a fast and compact algorithm.

The characters **$**, *****, **[**, ^, |, (,), and \ are interpreted literally by *fgrep*, that is, *fgrep* does not recognize full regular expressions as does *egrep*. Since these characters have special meaning to the shell, it is safest to enclose the entire *string* in single quotes '...'.

If no files are specified, *fgrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-b	Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
–c	Print only a count of the lines that contain the pattern.
-i	Ignore upper/lower case distinction during comparisons.
-l	Print the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.
-n	Precede each line by its line number in the file (first line is 1).
–v	Print all lines except those that contain the pattern.
–x	Print only lines matched entirely.
-e *special_string*	Search for a *special string* (*string* begins with a -).
-f *file*	Take the list of *strings* from *file*.

SEE ALSO

ed(1), egrep(1), grep(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* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.

NAME

file – determine file type

SYNOPSIS

file [**-c**] [**-f** ffile] [**-m** mfile] arg ...

DESCRIPTION

The *file* command 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.

-c The –c option 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.**

-f If the **-f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

-m The **-m** option instructs *file* to use an alternate magic file.

The *file* command 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.

FILES

/etc/magic

SEE ALSO

filehdr(4) in the *Programmer's Reference Manual.*

NAME

find – find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

The *find* command recursively descends the directory hierarchy for each path name in the *path-name-list* (that is, 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*. Valid expressions are:

-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, only the bits that are set in *onum* are compared with the file permission flags, and the expression evaluates true if they match.
-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 respectively.
-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*[c]	True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.
-atime *n*	True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find* itself.
-mtime *n*	True if the file has been modified in *n* days.
-ctime *n*	True if the file has been changed in *n* days.
-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*	Always true; write the current file on *device* in *cpio* (1) format (5120-byte records).
-newer *file*	True if the current file has been modified more recently than the argument *file*.
-depth	Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files that are contained in directories without write permission.
-mount	Always true; restricts the search to the file system containing the directory specified, or if no directory was specified, the current directory.
-local	True if the file physically resides on the local system.
(*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

chmod(1), cpio(1), sh(1), test(1).
stat(2), umask(2), fs(4) in the *Programmer's Reference Manual.*

BUGS

find / -depth always fails with the message: "find: stat failed: : No such file or directory".

NAME

format – format floppy disk tracks

SYNOPSIS

/bin/format [**-vVE**] [**-f** first] [**-l** last] [**-i** interleave] device[t]

DESCRIPTION

The *format* command formats floppy disks. Unless otherwise specified, formatting starts at track 0 and continues until an error is returned at the end of a partition.

The **-f** and **-l** options specify the first and last track to be formatted. The default interleave of 4 may be modified by using the **-i** option. *Device* must specify a raw (character) floppy device. The **t** indicates the entire disk. Absence of this letter indicates that the first track of the diskette cannot be accessed.

-v verbose.

-V verify. After tracks are formatted, a random sector is chosen and a write of test data is done into it. The sector is then read back and a compare is made.

-E exhausive verify. Every sector is verified by write/read/compare.

FILES

/dev/rdsk/* raw device for partition to be formatted

SEE ALSO

mkpart(1M), fd(7).

NAME

fsba – file system block analyzer

SYNOPSIS

/etc/fsba [**-b** target_block_size] file-system1 [file-system2 ...]

DESCRIPTION

The *fsba* command determines the number of additional 512-byte sectors required to store the data from an existing file system in a new file system with a different logical block size. Each *file-system* listed on the command line refers to an existing file system and should be specified by device name (e.g., **/dev/rdsk/0s2**).

The *target_block_size* specifies the logical block size in bytes of the new file system. Valid target block sizes are 512, 1024, and 2048. Default target block size is 1024. A block size of 2048 is supported only if the 2K file system package is installed.

The *fsba* command prints information about how many sectors are allocated to store the data in the old (existing) file system and how many would be required for a new file system with the specified logical block size. It also prints out the number of allocated and free i-nodes for the existing file system.

If the number of free sectors listed for the new file system is negative, the data will not fit in the new file system unless the new file system is larger than the existing file system. The new file system must be made at least as large as the number of sectors listed by *fsba* as allocated for the new file system. The maximum size of the new file system is limited by the disk partition used for the new file system.

Note that it is possible to specify a *target_block_size* that is smaller than the logical block size of the existing file system. In this case the new file system would require fewer sectors to store the data.

SEE ALSO

mkfs(1M).

NAME

fsck, dfsck – check and repair file systems

SYNOPSIS

/etc/fsck [**-y**] [**-n**] [**-s**X] [**-S**X] [**-t** file] [**-q**] [**-D**] [**-f**] [**-b**] [file-systems]
/etc/dfsck [options1] fsys1 ... **-** [options2] fsys2 ...

DESCRIPTION

fsck

The *fsck* command audits and interactively repairs inconsistent conditions for file systems. If the file system is found to be consistent, the number of files, blocks used, and blocks free are reported. If the file system is inconsistent, the user is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data loss may be determined from the diagnostic output. The default action for each correction is to wait for the user to respond **yes** or **no**. If the user does not have write permission, *fsck* defaults to a **-n** action.

The following options are accepted by *fsck*.

-y Assume a **yes** response to all questions asked by *fsck*.

-n Assume a **no** response to all questions asked by *fsck;* do not open the file system for writing.

-sX Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the super block will not continue to be used, or written on the file system.

The **-s**X option allows for creating an optimal free-list organization.

If X is not given, the values used when the file system was created are used. The format of X is *cylinder size:gap size.*

-SX Conditionally reconstruct the free list. This option is like **-s**X above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **-S** will force a "**no** response" to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

-t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **-t** flag, *fsck* will prompt the user for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

-q Quiet *fsck*. Do not print size-check messages. Unreferenced **fifos** will silently be removed. If *fsck* requires it, counts in the super block will be automatically fixed and the free list salvaged.

-D Directories are checked for bad blocks. Useful after system crashes.

-f Fast check. Check block and sizes and check the free list. The free list will be reconstructed if it is necessary.

-b Reboot. If the file system being checked is the root file system and modifications have been made, then either remount the root file system or reboot the system. A remount is done only if there was minor damage.

If no *file-systems* are specified, *fsck* will read a list of default file systems from the file **/etc/checklist**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Incorrect number of blocks.
   Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated i-node.
   I-node number out of range.
8. Super Block checks:
   More than 65536 i-nodes.
   More blocks for i-nodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the user's concurrence, reconnected by placing them in the **lost+found** directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. Empty files or directories are removed, as long as the **-n** option is not specified. *fsck* will force the reconnection of nonempty directories. The name assigned is the i-node number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster and should be used with everything but the *root* file system.

dfsck

The *dfsck* command allows two file system checks on two different drives simultaneously. *options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A – is the separator between the file system groups.

The *dfsck* command permits a user to interact with two *fsck* programs at once. To aid in this, *dfsck* will print the file system name for each message to the user. When answering a question from *dfsck*, the user must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

FILES

/etc/checklist	contains default list of file systems to check.

SEE ALSO

mkfs(1M), ncheck(1M), crash(1M).
uadmin(2), checklist(4), fs(4) in the *Programmer's Reference Manual.*

BUGS

I-node numbers for **.** and **..** in each directory are not checked for validity.

NAME

fsdb – file system debugger

SYNOPSIS

/etc/fsdb special [-]

DESCRIPTION

The *fsdb* command can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

The *fsdb* command contains several error-checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking *fsdb* with the optional - argument or by the use of the **O** symbol. (*fsdb* reads the i-size and f-size entries from the super block of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

The *fsdb* command reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i-number to i-node address
b	convert to block address
d	directory slot offset
+,-	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=-	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs, and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry, or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words, and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. I-nodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 – 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.
ln=4	changes the link count for the working i-node to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i-node.

2i.fd	prints the first 32 directory entries for the root i-node of this file system.
d5i.fc	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
512B.p0o	prints the super block of this file system in octal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.
a2b.p0d	prints the third block of the current i-node as directory entries.

SEE ALSO

fsck(1M).
dir(4), fs(4) in the *Programmer's Reference Manual.*

NAME

fsstat – report file system status

SYNOPSIS

/etc/fsstat special_file

DESCRIPTION

The *fsstat* command reports on the status of the file system on *special_file*. During startup, this command is used to determine if the file system needs checking before it is mounted. *fsstat* succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if the file system is active and not marked bad.

SEE ALSO

fs(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

The command has the following exit codes:

0 -- the file system is not mounted and appears okay,
(except for root where 0 means mounted and okay).
1 -- the file system is not mounted and needs to be checked.
2 -- the file system is mounted.
3 -- the command failed.

NAME

fstyp – determine file system identifier

SYNOPSIS

fstyp *special*

DESCRIPTION

The *fstyp* command allows the user to determine the file system identifier of mounted or unmounted file systems using heuristic programs. The file system type is required by *mount*(2) and sometimes by *mount*(1M) to mount file systems of different types.

The directory **/etc/fstyp.d** contains a program for each file system type to be checked; each of these programs applies some appropriate heuristic to determine whether the supplied *special* file is of the type for which it checks. If it is, the program prints on standard output the usual file-system identifier for that type and exits with a return code of 0; otherwise it prints error messages on standard error and exits with a non-zero return code. *fstyp* runs the programs in **/etc/fstyp.d** in alphabetical order, passing *special* as an argument; if any program succeeds, its file-system type identifier is printed and *fstyp* exits immediately. If no program succeeds, *fstyp* prints "Unknown_fstyp" to indicate failure.

WARNING

The use of heuristics implies that the result of *fstyp* is not guaranteed to be accurate.

SEE ALSO

mount(1M).
mount(2), sysfs(2) in the *Programmer's Reference Manual.*

NAME

fumount – forced unmount of an advertised resource

SYNOPSIS

fumount [–w *sec*] *resource*

DESCRIPTION

The **fumount** command unadvertises *resource* and disconnects remote access to the resource. The *–w sec* causes a delay of *sec* seconds prior to the execution of the disconnect.

When the forced unmount occurs, an administrative shell script is started on each remote computer that has the resource mounted (**/usr/bin/rfuadmin**). If a grace period of seconds is specified, **rfuadmin** is started with the **fuwarn** option. When the actual forced unmount is ready to occur, **rfuadmin** is started with the **fumount** option. See the **rfuadmin**(1M) man page for information on the action taken in response to the forced unmount.

This command is restricted to the super-user.

ERRORS

If *resource* (1) does not physically reside on the local machine, (2) is an invalid resource name, (3) is not currently advertised and is not remotely mounted, or (4) the command is not run with super-user privileges, an error message will be sent to standard error.

SEE ALSO

adv(1M), mount(1M), rfuadmin(1M), rfudaemon(1M), rmount(1M), unadv(1M).

NAME

fusage – disk access profiler

SYNOPSIS

fusage [[*mount_point*] | [*advertised_resource*] | [*block_special_device*] [...]]

DESCRIPTION

When used with no options, **fusage** reports block i/o transfers, in kilobytes, to and from all locally mounted file systems and advertised Remote File Sharing resources on a per client basis. The count data are cumulative since the time of the mount. When used with an option, **fusage** reports on the named file system, advertised resource, or block special device.

The report includes one section for each file system and advertised resource and has one entry for each machine that has the directory remotely mounted, ordered by decreasing usage. Sections are ordered by device name; advertised resources that are not complete file systems will immediately follow the sections for the file systems they are in.

SEE ALSO

adv(1M), mount(1M), df(1M), crash(1M).

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

/etc/fuser [**-ku**] files ¦ resources [-] [[**-ku**] files ¦ resources]

DESCRIPTION

The *fuser* command outputs the process IDs of the processes that are using the *files* or remote *resources* specified as arguments. Each process ID is followed by a letter code, interpreted as follows: if the process is using the file as 1. its current directory, the code is **c**; 2. the parent of its current directory (only when the file is being used by the system), the code is **p**; or 3. its root directory, the code is **r**. For block-special devices with mounted file systems, all processes using any file on that device are listed. For remote resource names, all processes using any file associated with that remote resource (Remote File Sharing) are reported. (**fuser** cannot use the mount point of the remote resource; it must use the resource name.) For all other types of files (text files, executables, directories, devices, etc.) only the processes using that file are reported.

The following options may be used with *fuser*:

-u the user login name, in parentheses, also follows the process ID.

-k the SIGKILL signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately [see *kill*(2)].

If more than one group of files are specified, the options may be respecified for each additional group of files. A lone dash cancels the options currently in force; then, the new set of options applies to the next group of files.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

You cannot list processes using a particular file from a remote resource mounted on your machine. You can only use the resource name as an argument.

Any user with permission to read **/dev/kmem** and **/dev/mem** can use **fuser**. Only the super-user can terminate another user's process

FILES

/unix	for system name list
/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

mount(1M), ps(1).
kill(2), signal(2) in the *Programmer's Reference Manual.*

NAME

fwtmp, wtmpfix – manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [**-ic**]
/usr/lib/acct/wtmpfix [files]

DESCRIPTION

fwtmp

fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in **wtmp** to formatted ASCII records. The ASCII version is useful to enable editing, via *ed*(1), bad records or general purpose maintenance of the file.

The argument **-ic** is used to denote that input is in ASCII form, and output is to be written in binary form.

wtmpfix

wtmpfix examines the standard input or named files in **wtmp** format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A - can be used in place of *files* to indicate the standard input. If time/date corrections are not performed, *acctcon*(1M) will fault when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to **/etc/wtmp**. The first record is the old date denoted by the string **old time** placed in the line field and the flag **OLD_TIME** placed in the type field of the **<utmp.h>** structure. The second record specifies the new date and is denoted by the string **new time** placed in the line field and the flag **NEW_TIME** placed in the type field. *wtmpfix* uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, *wtmpfix* will check the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to **INVALID** and write a diagnostic to the standard error. In this way, *wtmpfix* reduces the chance that *acctcon*(1M) will fail when processing connect accounting records.

FILES

/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), ed(1), runacct(1M).

acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

NAME

getopt – parse command options

SYNOPSIS

set -- 'getopt optstring **$*'**

DESCRIPTION

WARNING: Start using the new command *getopts*(1) in place of *getopt*(1). *getopt*(1) will not be supported in the next major release. For more information, see the **WARNINGS** section, below.

The *getopt* command 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 positional parameters ($1 $2 ...) of the shell 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

getopts(1), sh(1).
getopt(3C) in the *Programmer's Reference Manual.*

DIAGNOSTICS

The *getopt* command prints an error message on the standard error when it encounters an option letter not included in *optstring*.

WARNINGS

The *getopt*(1) command does not support the part of Rule 8 of the command syntax standard [see *intro*(1)] that permits groups of option-arguments following an option to be separated by white space and quoted. For example,

```
cmd -a -b -o "xxx z yy" file
```

is not handled correctly. To correct this deficiency, use the new command *getopts*(1) in place of *getopt*(1).

getopt(1) will not be supported in the next major release. For this release a conversion tool has been provided, *getoptcvt*. For more information about *getopts* and *getoptcvt*, see the *getopts*(1) manual page.

If an option that takes an option-argument is followed by a value that is the same as one of the options listed in *optstring* (referring to the earlier EXAMPLE section, but using the following command line: `cmd -o -a file`), *getopt* will always treat **-a** as an option-argument to **-o**; it will never recognize **-a** as an option. For this case, the **for** loop in the example will shift past the *file* argument.

NAME

getopts, getoptcvt – parse command options

SYNOPSIS

getopts optstring name [arg ...]

/usr/lib/getoptcvt [**-b**] file

DESCRIPTION

The *getopts* command is used by shell procedures to parse positional parameters and to check for legal options. It supports all applicable rules of the command syntax standard [see Rules 3-10, *intro*(1)]. It should be used in place of the *getopt*(1) command. (See the **WARNING**, below.)

optstring must contain the option letters the command using *getopts* will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, *getopts* will place the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell procedure is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, *getopts* places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name*.

When the end of options is encountered, *getopts* exits with a non-zero exit status. The special option "--" may be used to delimit the end of the options.

By default, *getopts* parses the positional parameters. If extra arguments (*arg* ...) are given on the *getopts* command line, *getopts* will parse them instead.

The */usr/lib/getoptcvt* command reads the shell script in *file*, converts it to use *getopts*(1) instead of *getopt*(1), and writes the results on the standard output.

-b the results of running */usr/lib/getoptcvt* will be portable to earlier releases of the UNIX system. */usr/lib/getoptcvt* modifies the shell script in *file* so that when the resulting shell script is executed, it determines at run time whether to invoke *getopts*(1) or *getopt*(1).

So all new commands will adhere to the command syntax standard described in *intro*(1), they should use *getopts*(1) or *getopt*(3C) to parse positional parameters and check for options that are legal for that command (see **WARNINGS**, below).

EXAMPLE

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
        case $c in
        a | b)      FLAG=$c;;
        o)          OARG=$OPTARG;;
        \?)         echo $USAGE
                    exit 2;;
        esac
done
shift `expr $OPTIND - 1`
```

This code will accept any of the following as equivalent:

```
cmd -a -b -o "xxx z yy" file
cmd -a -b -o "xxx z yy" -- file
cmd -ab -o xxx,z,yy file
cmd -ab -o "xxx z yy" file
cmd -o xxx,z,yy -b -a file
```

SEE ALSO

intro(1), sh(1).
getopt(3C) in the *Programmer's Reference Manual*.
UNIX System V Release 3.0 Release Notes.

WARNING

Although the following command syntax rule [see *intro*(1)] relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLE** section above, **a** and **b** are options, and the option **o** requires an option-argument:

`cmd -aboxxx file` (Rule 5 violation: options with option-arguments must not be grouped with other options)
`cmd -ab -oxxx file` (Rule 6 violation: there must be white space after an option that takes an option-argument)

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

DIAGNOSTICS

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

getty – set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**-h**] [**-t** timeout] line [speed [type [linedisc]]]
/etc/getty **-c** file

DESCRIPTION

The *getty* command is a program that is invoked by *init*(1M). It is the second process in the series, (*init-getty-login-shell*) that ultimately connects a user with the UNIX system. It can only be executed by the super-user; that is, a process with the user-ID of **root**. Initially *getty* prints the login message field for the entry it is using from **/etc/gettydefs**. *getty* reads the user's login name and invokes the *login*(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used. It does this by using the options and arguments specified.

Line is the name of a tty line in **/dev** to which *getty* is to attach itself. *getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. Unless *getty* is invoked with the **-h** flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The **-t** flag plus *timeout* (in seconds), specifies that *getty* should exit if the open on the line succeeds and no one types anything in the specified number of seconds.

Speed, the optional second argument, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* at what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by typing a <*break*> character). The default *speed* is 300 baud.

Type, the optional third argument, is a character string describing to *getty* what type of terminal is connected to the line in question. *getty* recognizes the following types:

none	default
ds40-1	DATASPEED terminal 40/1
tektronix,tek	TEKTRONIX
vt61	Digital Equipment vt61
vt100	Digital Equipment vt100
hp45	Hewlett-Packard 45
c100	Concept 100

The default terminal is **none**; i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition.

Linedisc, the optional fourth argument, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, **LDISC0**.

When given no optional arguments, *getty* sets the *speed* of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, new-line characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

After the user's name has been typed in, it is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately [see *ioctl*(2)].

The user's name is scanned to see if it contains any lower case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper case characters into the corresponding lower case characters.

Finally, *login* is **exec**'d with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which will place them in the environment [see *login*(1)].

A check option is provided. When *getty* is invoked with the **-c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs

SEE ALSO

ct(1C), init(1M), login(1), tty(7).
ioctl(2), gettydefs(4), inittab(4) in the *Programmer's Reference Manual.*

BUGS

While *getty* understands simple single character quoting conventions, it is not possible to quote certain special control characters used by *getty*. Thus, you cannot log in via *getty* and type a **#**, **@**, **/**, **!**, **_**, backspace, **^U**, **^D**, or **&** as part of your login name or arguments. *getty* uses them to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. They will always be interpreted as having their special meaning.

NAME

graph – draw a graph

SYNOPSIS

graph [options]

DESCRIPTION

The *graph* command 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, do not erase before plotting.
-x [**l**]	If **l** 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 [**l**]	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

The *graph* command stores all points internally and drops those for which there is no room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

greek – select terminal filter

SYNOPSIS

greek [**-T**terminal]

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 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 **$TERM** [see *environ*(5)]. Currently, the following *terminal*s are recognized:

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), hp(1), tplot(1G).
environ(5), term(5) in the *Programmer's Reference Manual.*

NAME

grep – search a file for a pattern

SYNOPSIS

grep [options] limited regular expression [file ...]

DESCRIPTION

The *grep* command searches files for a pattern and prints all lines that contain that pattern. The *grep* command uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with *ed (1)* to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters **$**, *****, **[**, **^**, **|**, **(**, **)**, and \ in the *limited regular expression* because they are also meaningful to the shell. It is safest to enclose the entire *limited regular expression* in single quotes '...'.

If no files are specified, *grep* assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-b	Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
-c	Print only a count of the lines that contain the pattern.
-i	Ignore upper/lower case distinction during comparisons.
-l	Print the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.
-n	Precede each line by its line number in the file (first line is 1).
-s	Suppress error messages about nonexistent or unreadable files
-v	Print all lines except those that contain the pattern.

SEE ALSO

ed(1), egrep(1), fgrep(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

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.
If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

NAME

hp – handle special functions of Hewlett-Packard 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 usage is in conjunction with DOCUMENTER'S WORKBENCH Software:

```
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 *300*(1), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown.

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), greek(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

id – print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

The *id* command outputs the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

SEE ALSO

logname(1).
getuid(2) in the *Programmer's Reference Manual.*

NAME

idbuild – build new UNIX system kernel

SYNOPSIS

/etc/conf/bin/idbuild

DESCRIPTION

This script builds a new UNIX system kernel using the current system configuration in *etc/conf/*. Kernel reconfigurations are usually done after a device driver is installed, or system tunable parameters are modified. The script uses the shell variable *$ROOT* from the user's environment as its starting path. Except for the special case of kernel development in a non-root source tree, the shell variable *ROOT* should always be set to null, or to "/". *Idbuild* exits with a return code of zero on success and non-zero on failure.

Building a new UNIX system image consists of generating new system configuration files, then link-editing the kernel and device driver object modules in the *etc/conf/pack.d* object tree. This is done by *idbuild* by calling the following commands:

etc/conf/bin/idconfig	To build kernel configuration files.
etc/conf/bin/idmkunix	To process the configuration files and link-edit a new UNIX system image.

The system configuration files are built by processing the Master and System files representing device driver and tunable parameter specifications. For the i386 UNIX system the files *etc/conf/cf.d/mdevice,* and *etc/conf/cf.d/mtune* represent the Master information. The files *etc/conf/cf.d/stune,* and the files specified in *etc/conf/sdevice.d/** represent the System information. The kernel also has file system type information defined in the files specified by *etc/conf/sfsys.d/** and *etc/conf/mfsys.d/**

Once a new UNIX system kernel has been configured a lock file is set in *etc/.new_unix* which causes the new kernel to replace */unix* on the next system shutdown (i.e. on the next entry to the *init 0* state). Upon the next system boot the new kernel will be executed.

ERROR MESSAGES

Since *idbuild* calls other system commands to accomplish system reconfiguration and link editing, it will report all errors encountered by those commands, then clean up intermediate files created in the process. In general, the exit value 1 indicates an error was encountered by *idbuild*.

The errors encountered fall into the following categories:

Master file error messages.
System file error messages.
Tunable file error messages.
Compiler and Link-editor error messages.

All error messages are designed to be self-explanatory.

SEE ALSO

idinstall(1m), idtune(1m).
mdevice(4), mfsys(4), mtune(4), sdevice(4), sfsys(4), stune(4) in the *Programmer's Reference Manual.*

NAME

idcheck – returns selected information

SYNOPSIS

/etc/conf/bin/idcheck

DESCRIPTION

This command returns selected information about the system configuration. It is useful in add-on device Driver Software Package (DSP) installation scripts to determine if a particular device driver has already been installed, or to verify that a particular interrupt vector, I/O address or other selectable parameter is in fact available for use. The various forms are:

idcheck –p device-name [*–i dir*] [*–r*]

idcheck –v vector [*–i dir*] [*–r*]

idcheck –d dma-channel [*–i dir*] [*–r*]

idcheck –a –l lower_address –u upper_address [*–i dir*] [*–r*]

idcheck –c –l lower_address –u upper_address [*–i dir*] [*–r*]

This command scans the System and Master modules and returns:

100, if an error occurs.

0, if no conflict exists.

a positive number greater than 0 and less than 100 if a conflict exists.

The command line options are:

–r	Report device name of any conflicting device on stdout.
–p device-name	This option checks for the existence of four different components of the DSP. The exit code is the addition of the return codes from the four checks. Add 1 to the exit code if the DSP directory under */etc/conf/pack.d* exists. Add 2 to the exit code if the Master module has been installed. Add 4 to the exit code if the System module has been installed. Add 8 to the exit code if the Kernel was built with the System module. Add 16 to the exit code if a Driver.o is part of the DSP (vs. a stubs.c file).
–v vector	Returns 'type' field of device that is using the vector specified (i.e. Another DSP is already using the vector).
–d dma-channel	Returns 1 if the dma channel specified is being used.
–a	This option checks whether the IOA region bounded by "lower" and "upper" conflict with another DSP ("lower" and "upper" are specified with the **–l** and **–u**

options). The exit code is the addition of two different return codes.

Add 1 to the exit code if the IOA region overlaps with another device.

Add 2 to the exit code if the IOA region overlaps with another device and that device has the 'O' option specified in the *type* field of the Master module. The 'O' option permits a driver to overlap the IOA region of another driver.

–c Returns 1 if the CMA region bounded by "lower" and "upper" conflict with another DSP ("lower" and "upper" are specified with the **–l** and **–u** options).

–l address Lower bound of address range specified in hex. The leading 0x is unnecessary.

–u address Upper bound of address range specified in hex. The leading 0x is unnecessary.

–i dir Specifies the directory in which the ID files *sdevice* and *mdevice* reside. The default directory is */etc/conf/cf.d*.

ERROR MESSAGES

There are no error messages or checks for valid arguments to options. *Idcheck* interprets these arguments using the rules of *scanf*(3) and queries the *sdevice,* and *mdevice* files. For example, if a letter is used in the place of a digit, *scanf*(3) will translate the letter to 0. *Idcheck* will then use this value in its query.

SEE ALSO

idinstall(1m).

mdevice(4), sdevice(4) in the *Programmer's Reference Manual.*

NAME

idinstall – add, delete, update, or get device driver configuration data

SYNOPSIS

/etc/conf/bin/idinstall –[**adug**] [**–e**] –[**msoptnirhcl**] dev_name

DESCRIPTION

The *idinstall* command is called by a Driver Software Package (DSP) Install script or Remove script to Add (–a), Delete (–d), Update (–u), or Get (–g) device driver configuration data. *Idinstall* expects to find driver component files in the current directory. When components are installed or updated they are moved or appended to files in the */etc/conf* directory and then deleted from the current directory unless the *-k* flag is used. The options for the command are as follows:

Action Specifiers:

–a	Add the DSP components
–d	Remove the DSP components
–u	Update the DSP components
–g	Get the DSP components (print to std out, except Master)

Component Specifiers: (*)

–m	Master component
–s	System component
–o	Driver.o component
–p	Space.c component
–t	Stubs.c component
–n	Node (special file) component
–i	Inittab component
–r	Device Initialization (rc) component
–h	Device shutdown (sd) component
–c	Mfsys component: file system type config (Master) data.
–l	Sfsys component: file system type local (System) data.

(*) If no component is specified the default is all except for the –g option where a single component must be specified explicitly.

Miscellaneous:

–e	Disable free disk space check
–k	Keep files (do not remove from current directory) on add or update.

In the simplest case of installing a new DSP the command syntax used by the DSP's *Install* script should be *idinstall -a dev_name* . In this case the command will require and install a Driver.o, Master and System entry, and optionally install the Space.c, Stubs.c, Node, Init, Rc, Shutdown, Mfsys, and Sfsys components if those modules are present in the current directory.

The Driver.o, Space.c, and Stubs.c files are moved to a directory in */etc/conf/pack.d*. The *dev_name* is passed as an argument, which is used as the directory name. The remaining components are stored in the corresponding directories under */etc/conf* in a file whose name is *dev_name*. For example, the Node file would be moved to */etc/conf/node.d/dev_name*.

The *idinstall -m* usage provides an interface to the *idmaster* command which will add, delete and update *mdevice* file entries using a Master file from the local directory. An interface is provided here so that driver writers have a consistent interface to install any DSP component.

As stated above, driver writers will generally use only the *idinstall -a dev_name* form of the command. Other options of *idinstall* are provided to allow an Update DSP (i.e. one that replaces an existing device driver component) to be installed, and to support installation of multiple controller boards of the same type.

If the call to *idinstall* uses the *-u* (update) option, it will:

> overlay the files of the old DSP with the files of the new DSP.
>
> invoke the *idmaster* command with the 'update' option if a Master module is part of the new DSP.

Idinstall also does a verification that enough free disk space is available to start the reconfiguration process. This is done by calling the *idspace* command. *Idinstall* will fail if insufficient space exists, and exit with a non-zero return code. The *-e* option bypasses this check.

Idinstall makes a record of the last device installed in a file *(/etc/.last_dev_add)* , and saves all removed files from the last delete operation in a directory *(/etc/.last_dev_del)* . These files are recovered by */etc/conf/bin/idmkenv* whenever it is determined that a system reconfiguration was aborted due to a power failure or unexpected system reboot.

ERROR MESSAGES

An exit value of zero indicates success. If an error was encountered *idinstall* will exit with a non-zero value, and report an error message. All error messages are designed to be self-explanatory. Typical error message that can be generated by *idinstall* are as follows:

> *Device package already exists.*
> *Cannot make the driver package directory.*
> *Cannot remove driver package directory*
> *Local directory does not contain a Driver object (Driver.o) file.*
> *Local directory does not contain a Master file.*
> *Local directory does not contain a System file.*
> *Cannot remove driver entry.*

SEE ALSO

idspace(1m), idcheck(1m).
mdevice(4), sdevice(4) in the *Programmer's Reference Manual.*

NAME

idload – Remote File Sharing user and group mapping

SYNOPSIS

idload [**-n**] [**-g** *g_rules*] [**-u** *u_rules*] [*directory*]
idload -k

DESCRIPTION

idload is used on Remote File Sharing server machines to build translation tables for user and group ids. It takes your **/etc/passwd** and **/etc/group** files and produces translation tables for user and group ids from remote machines, according to the rules set down in the *u_rules* and *g_rules* files. If you are mapping by user and group name, you will need copies of remote **/etc/passwd** and **/etc/group** files. If no rules files are specified, remote user and group ids are mapped to MAXUID+1 (this is an id number that is one higher than the highest number you could assign on your system.)

By default, the remote password and group files are assumed to reside in **/usr/nserve/auth.info/***domain/nodename/*[**passwd | group**]. The *directory* argument indicates that some directory structure other than **/usr/nserve/auth.info** contains the *domain/nodename* **passwd** and **group** files. (*nodename* is the name of the computer the files are from and *domain* is the domain that computer is a member.)

You must run **idload** to put the mapping into place. Global mapping will take effect immediately for machines that have one of your resources currently mounted. Mapping for other specific machines will take effect when each machine mounts one of your resources.

-n	This is used to do a trial run of the id mapping. No translation table will be produced; however, a display of the mapping is output to the terminal (*stdout*).
-k	This is used to print the idmapping that is currently in use. (Specific mapping for remote machines will not be shown until that machine mounts one of your resources.)
-u *u_rules*	The *u_rules* file contains the rules for user id translation. The default rules file is **/usr/nserve/auth.info/uid.rules**.
-g *g_rules*	The *g_rules* file contains the rules for group id translation. The default rules file is **/usr/nserve/auth.info/gid.rules**.

This command is restricted to the super-user.

Rules

The rules files have two types of sections (both optional): **global** and **host**. There can be only one global section, though there can be one host section for each computer you want to map.

The **global** section describes the default conditions for translation for any machines that are not explicitly referenced in a **host** section. If the global section is missing, the default action is to map all remote user and group ids from undefined computers to MAXUID+1. The syntax of the first line of the **global** section is:

global

A **host** section is used for each machine or group of machines that you want to map differently from the global definitions. The syntax of the first line of each **host** section is:

host *name ...*

where *name* is replaced by the full name of a computer (*domain.nodename*).

The format of a rules file is described below. (All lines are optional, but must appear in the order shown.)

global
default *local* | **transparent**
exclude *remote_id-remote_id* | *remote_id*
map *remote_id:local*

host *domain.nodename* [*domain.nodename...*]
default *local* | **transparent**
exclude *remote_id-remote_id* | *remote_id* | *remote_name*
map *remote:local* | *remote* | **all**

Each of these instruction types is described below.

The line

default *local* | **transparent**

defines the mode of mapping for remote users that are not specifically mapped in instructions in other lines. **transparent** means that each remote user and group id will have the same numeric value locally unless it appears in the **exclude** instruction. *local* can be replaced by a local user name or id to map all users into a particular local name or id number. If the default line is omitted, all users that are not specifically mapped are mapped into a "special guest" login id.

The line

exclude *remote_id-remote_id* | *remote_id* | *remote_name*

defines remote ids that will be excluded from the **default** mapping. The **exclude** instruction must precede any **map** instructions in a block. You can use a range of id numbers, a single id number, or a single name. (*remote_name* cannot be used in a *global* block.)

The line

map *remote:local* | *remote* | **all**

defines the local ids and names that remote ids and names will be mapped into. *remote* is either a remote id number or remote name; *local* is either a local id number or local name. Placing a colon between a *remote* and a *local* will give the value on the left the permissions of the value on the right. A single *remote* name or id will assign the user or group permissions of the same local name or id. **all** is a predefined alias for the set of all user and group ids found in the local **/etc/passwd** and **/etc/group** files. (You cannot map by remote name in **global** blocks.)

NOTE: **idload** will always output warning messages for **map all**, since password files always contain multiple administrative user names with the same id number. The first mapping attempt on the id number will succeed; each subsequent attempt will produce a warning.

Remote File Sharing doesn't need to be running to use **idload**.

EXIT STATUS

On successful completion, **idload** will produce one or more translation tables and return a successful exit status. If **idload** fails, the command will return an exit status of zero and not produce a translation table.

ERRORS

If (1) either rules file cannot be found or opened; (2) there are syntax errors in the rules file; (3) there are semantic errors in the rules file; (4) **host** password or group information could not be found; or (5) the command is not run with super-user privileges, an error message will be sent to standard error. Partial failures will cause a warning message to appear, though the process will continue.

FILES

/etc/passwd
/etc/group
/usr/nserve/auth.info/*domain/nodename*/[**user | group**]
/usr/nserve/auth.info/uid.rules
/usr/nserve/auth.info/gid.rules

SEE ALSO

mount(1M).
"Remote File Sharing" chapter of the *System Administrator's Guide* for detailed information on ID mapping.

NAME

idmkinit – reads files containing specifications

SYNOPSIS

/etc/conf/bin/idmkinit

DESCRIPTION

This command reads the files containing specifications of */etc/inittab* entries from */etc/conf/init.d* and constructs a new *inittab* file in */etc/conf/cf.d.* It returns 0 on success and a positive number on error.

The files in */etc/conf/init.d* are copies of the Init modules in device Driver Software Packages (DSP). There is at most one Init file per DSP. Each file contains one line for each *inittab* entry to be installed. There may be multiple lines (i.e. multiple *inittab* entries) per file. An *inittab* entry has the form

id:rstate:action:process.

The Init module entry must have either of the following forms:

action:process or *rstate:action:process.*

When *idmkinit* encounters an entry of the first type, a valid *id* field will be generated, and an *rstate* field of 2 (indicating run on init state 2) will be generated. When a entry of the second type is encountered only the *id* field is prepended.

The *idmkinit* command is called automatically upon entering init state 2 on the next system reboot after a kernel reconfiguration to establish the correct */etc/inittab* for the running */unix* kernel. *Idmkinit* can be called as a user level command to test modification of *inittab* before a DSP is actually built. It is also useful by installation scripts that do not reconfigure the kernel, but need to create *inittab* entries. In this case the *inittab* generated by *idmkinit* must be copied to */etc/inittab,* and a *telinit q* command must be run to make the new entry take affect.

The command line options are:

–o directory	*Inittab* will be installed in the directory specified rather than */etc/conf/cf.d.*
–i directory	The ID file "init.base", which normally resides in */etc/conf/cf.d,* can be found in the directory specified.
–e directory	The Init modules that are usually in */etc/conf/init.d* can be found in the directory specified.
–#	Print debugging information.

ERROR MESSAGES

An exit value of zero indicates success. If an error was encountered, *idmkinit* will exit with a non-zero value, and report an error message. All error messages are designed to be self-explanatory.

SEE ALSO

idbuild(1), idinstall(1m), idmknod(1m), init(1m).

NAME

idmknod – removes nodes and reads specifications of nodes

SYNOPSIS

/etc/conf/bin/idmknod

DESCRIPTION

This command performs the following functions:

Removes the nodes for non-required devices (those that do not have an 'r' in field 3 of the the device's *mdevice* entry) from */dev*. Ordinary files will not be removed. If the */dev* directory contains subdirectories, those subdirectories will be transversed and nodes found for non-required devices will be removed as well. If empty subdirectories result due to the removal of nodes, the subdirectories are then removed.

Reads the specifications of nodes given in the files contained in */etc/conf/node.d* and installs these nodes in */dev*. If the node specification defines a path containing subdirectories, the subdirectories will be made automatically.

Returns 0 on success and a positive number on error.

The *idmknod* command is run automatically upon entering init state 2 on the next system reboot after a kernel reconfiguration to establish the correct representation of device nodes in the */dev* directory for the running */unix* kernel. *idmknod* can be called as a user level command to test modification of the */dev* directory before a DSP is actually built. It is also useful in installation scripts that do not reconfigure the kernel, but need to create */dev* entries.

The files in */etc/conf/node.d* are copies of the *Node* modules installed by device Driver Software Packages (DSP). There is at most one file per DSP. Each file contains one line for each node that is to be installed. The format of each line is:

Name of device entry (field 1) in the *mdevice* file (The *mdevice* entry will be the line installed by the DSP from its *Master* module). This field must be from 1 to 8 characters in length. The first character must be a letter. The others may be letters, digits, or underscores.

Name of node to be inserted in /dev. The first character must be a letter. The others may be letters, digits, or underscores. This field can be a path relative to */dev*, and *idmknod* will create subdirectories as needed.

The character 'b' or 'c'. A 'b' indicates that the node is a 'block' type device and 'c' indicates 'character' type device.

Minor device number. This value must be between 0 and 255. If this field is a non-numeric, it is assumed to be a request for a streams clone device node, and *idmknod* will set the minor number to the value of the major number of the device specified.

Some example node file entries are as follows:

asy tty00 c 1	makes /dev/tty00 for device 'asy' using minor device 1.
qt rmt/c0s0 c 4	makes /dev/rmt/c0s0 for device 'qt' using minor device 4.
clone net/nau/clone c nau	makes /dev/net/nau/clone for device 'clone'. The minor device number is set to the major device number of device 'nau'.

The command line options are:

-o directory	Nodes will be installed in the directory specified rather than */dev*.
-i directory	The file "mdevice" which normally resides in */etc/conf/cf.d,* can be found in the directory specified.
-e directory	The *Node* modules that normally reside in */etc/conf/node.d* can be found in the directory specified.
-s	Suppress removing nodes (just add new nodes).

ERROR MESSAGES

An exit value of zero indicates success. If an error was encountered due to a syntax or format error in a *node* entry an advisory message will be printed to *stdout* and the command will continue. If a serious error is encountered (i.e. a required file can not be found) *idmknod* will exit with a non-zero value, and report an error message. All error messages are designed to be self-explanatory.

SEE ALSO

idinstall(1m), idmkinit(1m).

mdevice(4), sdevice(4) in the *Programmer's Reference Manual.*

NAME

idspace – investigates free space

SYNOPSIS

/etc/conf/bin/idspace [**-i** inodes] [**-r** blocks] [**-u** blocks]
[**-t** blocks]

DESCRIPTION

This command investigates free space in /, **/usr**, and **/tmp** file systems to determine whether sufficient disk blocks and inodes exist in each of potentially 3 file systems. The default tests that *idspace* performs are as follows:

Verify that the root file system (/) has 400 blocks more than the size of the current **/unix**. This verifies that a device driver being added to the current **/unix** can be built and placed in the root directory. A check is also made to insure that 100 inodes exist in the root directory.

Determine whether a **/usr** file system exists. If it does exist a test is made that 400 free blocks and 100 inodes are available in that file system. If the file system does not exist *idspace* does not complain since files created in **/usr** by the reconfiguration process will be created in the root file system and space requirements are covered by the test in (1.) above.

Determine whether a **/tmp** file system exists. If it does exist a test is made that 400 free blocks and 100 inodes are available in that file system. If the file system does not exist *idspace* does not complain since files created in **/tmp** by the reconfiguration process will be created in the root file system and space requirements are covered by the test in (1.) above.

The command line options are:

–i inodes	This option overrides the default test for 100 inode in all of the *idspace* checks.
–r blocks	This option overrides the default test for */unix* size + 400 blocks when checking the root *(/)* file system. When the **-r** option is used the **/usr** and **/tmp** file systems are not tested unless also explicitly specified.
–u blocks	This option overrides the default test for 400 blocks when checking the **/usr** file system. When the **-u** option is used the root (/) and **/tmp** file systems are not tested unless also explicitly specified. If **/usr** is not a separate file system an error is reported.
–t blocks	This option overrides the default test for 400 blocks when checking the **/tmp** file system. When the **-t** option is used the root (/) and **/usr** file systems are not tested unless also explicitly specified. If **/tmp** is not a separate file system an error is reported.

ERROR MESSAGES

An exit value of zero indicates success. If insufficient space exists in a file system or an error was encountered due to a syntax or format error *idspace* will report a message. All error messages are designed to be self-explanatory. The specific exit values are as follows:

0 success.

1 command syntax error, or needed file does not exist.

2 file system has insufficient space or inodes.

3 requested file system does not exist (**-u** and **-t** options only).

SEE ALSO

idbuild(1m), idinstall(1m).

NAME

idtune – attempts to set value of a tunable parameter

SYNOPSIS

/etc/conf/bin/idtune [**-f** | **-m**] name value

DESCRIPTION

This script attempts to set the value of a tunable parameter. The tunable parameter to be changed is indicated by *name*. The desired value for the tunable parameter is *value*.

If there is already a value for this parameter (in the **stune** file), the user will normally be asked to confirm the change with the following message:

> Tunable Parameter *name* is currently set to *old_value*.
> Is it OK to change it to *value*? (y/n)

If the user answers 'y', the change will be made. Otherwise, the tunable parameter will not be changed, and the following message will be displayed:

> *name* left at *old_value*.

However, if the **-f** (force) option is used, the change will always be made, and no messages will ever be given.

If the **-m** (minimum) option is used and there is an existing value which is greater than the desired value, no change will be made and no message will be given.

If system tunable parameters are being modified as part of a device driver or application add-on package, it may not be desirable to prompt the user with the above question. The add-on package Install script may chose to override the existing value using the **-f** or **-m** options. However, care must be taken not to invalidate a tunable parameter modified earlier by the user or another add-on package.

In order for the change in parameter to become effective, the UNIX system kernel must be rebuilt, and the system rebooted.

DIAGNOSTICS

The exit status will ne non-zero if errors are encountered.

SEE ALSO

idbuild(1).

mtune(4), stune(4) in the *Programmer's Reference Manual.*

NAME

infocmp – compare or print out terminfo descriptions

SYNOPSIS

infocmp [**-d**] [**-c**] [**-n**] [**-I**] [**-L**] [**-C**] [**-r**] [**-u**] [**-s d|i|l|c**] [**-v**] [**-V**] [**-1**] [**-w** width] [**-A** directory] [**-B** directory] [termname ...]

DESCRIPTION

The *infocmp* command can be used to compare a binary *terminfo*(4) entry with other terminfo entries, rewrite a *terminfo*(4) description to take advantage of the **use=** terminfo field, or print out a *terminfo*(4) description from the binary file [*term*(4)] in a variety of formats. In all cases, the Boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default Options

If no options are specified and zero or one *termnames* are specified, the **-I** option will be assumed. If more than one *termname* is specified, the **-d** option will be assumed.

Comparison Options [–d] [–c] [–n]

The *infocmp* command compares the *terminfo*(4) description of the first terminal *termname* with each of the descriptions given by the entries for the other terminal's *termnames*. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: **F** for boolean variables, **-1** for integer variables, and **NULL** for string variables.

-d produce a list of each capability that is different. In this manner, if one has two entries for the same terminal or similar terminals, using *infocmp* will show what is different between the two entries. This is sometimes necessary when more than one person produces an entry for the same terminal and one wants to see what is different between the two.

-c produce a list of each capability that is common between the two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **-u** option is worth using.

-n produce a list of each capability that is in neither entry. If no *termnames* are given, the environment variable **TERM** will be used for both of the *termnames*. This can be used as a quick check to see if anything was left out of the description.

Source Listing Options [–I] [–L] [–C] [–r]

The **-I**, **-L**, and **-C** options will produce a source listing for each terminal named.

-I use the *terminfo*(4) names

-L use the long C variable name listed in <**term.h**>

-C use the *termcap* names

-r when using **-C**, put out all capabilities in *termcap* form

If no *termnames* are given, the environment variable **TERM** will be used for the terminal name.

The source produced by the **-C** option may be used directly as a *termcap* entry, but not all of the parameterized strings may be changed to the *termcap* format. *infocmp* will attempt to convert most of the parameterized information, but that which it doesn't will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where *termcap* expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All *termcap* variables no longer supported by *terminfo*(4), but which are derivable from other *terminfo*(4) variables, will be output. Not all *terminfo*(4) capabilities will be translated; only those variables which were part of *termcap* will normally be output. Specifying the **-r** option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output, mandatory padding is not supported, and *termcap* strings were not as flexible, it is not always possible to convert a *terminfo*(4) string capability into an equivalent *termcap* format. Not all of these strings will be able to be converted. A subsequent conversion of the *termcap* file back into *terminfo*(4) format will not necessarily reproduce the original *terminfo*(4) source.

Some common *terminfo* parameter sequences, their *termcap* equivalents, and some terminal types which commonly have such sequences, are:

Terminfo	Termcap	Representative Terminals
%p1%c	%.	adm
%p1%d	%d	hp, ANSI standard, vt100
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	hp

Use= Option [-u]

-u produce a *terminfo*(4) source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals *termnames*. It does this by analyzing the differences between the first *termname* and the other *termnames* and producing a description with **use**= fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using *infocmp* will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname*, but one of the other *termname* entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries that has this capability gives a different value for the capability than that in the first *termname*.

The order of the other *termname* entries is significant. Since the terminfo compiler **tic**(1M) does a left-to-right scan of the capabilities, specifying two **use**= entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given in. *infocmp* will flag any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability *after* a **use**= entry that contains that capability will cause the second specification to be ignored. Using *infocmp* to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra **use**= fields that are superfluous. *infocmp* will flag any other *termname* **use**= fields that were not needed.

Other Options [-s d|i|l|c] [-v] [-V] [-1] [-w width]

-s sort the fields within each type according to the argument below:

- **d** leave fields in the order that they are stored in the *terminfo* data base.
- **i** sort by *terminfo* name.
- **l** sort by the long C variable name.
- **c** sort by the *termcap* name.

If no **-s** option is given, the fields printed out will be sorted alphabetically by the *terminfo* name within each type, except in the case of the **-C** or the **-L** options, which cause the sorting to be done by the *termcap* name or the long C variable name, respectively.

-v print out tracing information on standard error as the program runs.

-V print out the version of the program in use on standard error and exit.

-1 cause the fields to print out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

-w change the output to width characters.

Changing Data Bases [-A directory] [-B directory]

The location of the compiled *terminfo*(4) data base is taken from the environment variable **TERMINFO**. If the variable is not defined, or the terminal is not found in that location, the system *terminfo*(4) data base, usually in */usr/lib/terminfo*, will be used. The options **-A** and **-B** may be used to override this location. The **-A** option will set **TERMINFO** for the first *termname* and the **-B** option will set **TERMINFO** for the other *termnames*. With this, it is possible to compare descriptions for a terminal with the same name located in two different data bases. This is useful for comparing descriptions for the same terminal created by different people. Otherwise the terminals would have to be named differently in the *terminfo*(4) data base for a comparison to be made.

FILES

/usr/lib/terminfo/?/* compiled terminal description data base

DIAGNOSTICS

malloc is out of space!
: There was not enough memory available to process all the terminal descriptions requested. Run *infocmp* several times, each time including a subset of the desired *termnames*.

use= order dependency found:
: A value specified in one relative terminal specification was different from that in another relative terminal specification.

'use=*term*' did not add anything to the description.
: A relative terminal name did not contribute anything to the final description.

must have at least two terminal names for a comparison to be done.
: The **-u**, **-d**, and **-c** options require at least two terminal names.

SEE ALSO

captoinfo(1M).
tic(1M), curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual*.
Chapter 10 of the *Programmer's Guide*.

NOTE

The *termcap* data base (from earlier releases of UNIX System V) may not be supplied in future releases.

NAME

init, telinit – process control initialization

SYNOPSIS

/etc/init [0123456SsQqabc]

/etc/telinit [0123456SsQqabc]

DESCRIPTION

Init

init is a general process spawner. Its primary role is to create processes from information stored in the file **/etc/inittab** [see *inittab*(4)].

At any given time, the system is in one of eight possible run levels. A run level is a software configuration of the system under which only a selected group of processes exist. The processes spawned by *init* for each of these run levels is defined in **/etc/inittab**. *init* can be in one of eight run levels, **0–6** and **S** or **s** (run levels **S** and **s** are identical). The run level changes when a privileged user runs **/etc/init**. This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was booted, telling it which run level to change to.

The following are the arguments to *init*.

0	shut the machine down so it is safe to remove the power. Have the machine remove power if it can. This state can be executed only from the console.
1	put the system in single-user mode. Unmount all file systems except root. All user processes are killed except those connected to the console. This state can be executed only from the console.
2	put the system in multi-user mode. All multi-user environment terminal processes and daemons are spawned. This state is commonly referred to as the multi-user state.
3	start the remote file sharing processes and daemons. Mount and advertise remote resources. Run level **3** extends multi-user mode and is known as the remote-file-sharing state.
4	is available to be defined as an alternative multi-user environment configuration. It is not necessary for system operation and is usually not used.
5	Stop the UNIX system and go to the firmware monitor.
6	Stop the UNIX system and reboot to the state defined by the `initdefault` entry in **/etc/inittab**.
a,b,c	process only those **/etc/inittab** entries having the **a**, **b** or **c** run level set. These are pseudo-states, which may be defined to run certain commands, but which do not cause the current run level to change.
Q,q	re-examine **/etc/inittab**.

S,s enter single-user mode. When this occurs, the terminal which executed this command becomes the system console. This is the only run level that doesn't require the existence of a properly formatted **/etc/inittab** file. If this file does not exist, then by default the only legal run level that *init* can enter is the single-user mode. When the system enters **S** or **s**, all mounted file systems remain mounted and only processes spawned by init are killed.

When a UNIX system is booted, *init* is invoked and the following occurs. First, *init* looks in **/etc/inittab** for the `initdefault` entry [see **inittab**(4)]. If there is one, *init* uses the run level specified in that entry as the initial run level to enter. If there is no `initdefault` entry in **/etc/inittab,** *init* requests that the user enter a run level from the virtual system console. If an **S** or **s** is entered, *init* goes to the single-user state. In the single-user state the virtual console terminal is assigned to the user's terminal and is opened for reading and writing. The command **/bin/su** is invoked and a message is generated on the physical console saying where the virtual console has been relocated. Use either *init* or *telinit,* to signal *init* to change the run level of the system. Note that if the shell is terminated (via an end-of-file), *init* will only re-initialize to the single-user state if the **/etc/inittab** file does not exist.

If a **0** through **6** is entered, *init* enters the corresponding run level. Note that, on the 80386 computer, the run levels **0**, **1**, **5**, and **6** are reserved states for shutting the system down; the run levels **2**, **3**, and **4** are available as normal operating states.

On your computer, the *run-levels* **0** and **1** are reserved states for shutting the system down, and *run-levels* **2, 3,** and **4** are available as normal operating states.

If this is the first time since power up that *init* has entered a run level other than single-user state, *init* first scans **/etc/inittab** for `boot` and `bootwait` entries [see *inittab*(4)]. These entries are performed before any other processing of **/etc/inittab** takes place, providing that the run level entered matches that of the entry. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. *init* then scans **/etc/inittab** and executes all other entries that are to be processed for that run level.

In a multi-user environment, **/etc/inittab** is set up so that *init* will create a *getty* process for each terminal that the administrator sets up to respawn.

To spawn each process in **/etc/inittab,** *init* reads each entry and for each entry that should be respawned, it forks a child process. After it has spawned all of the processes specified by **/etc/inittab,** *init* waits for one of its descendant processes to die, a powerfail signal, or a signal from another *init* or *telinit* process to change the system's run level. When one of these conditions occurs, *init* re-examines **/etc/inittab.** New entries can be added to **/etc/inittab** at any time; however, *init* still waits for one of the above three conditions to occur before re-examining **/etc/inittab**. To get around this, **init Q** or **init q** command wakes *init* to re-examine **/etc/inittab** immediately.

When *init* comes up at boot time and whenever the system changes from the single-user state to another run state, init sets the *ioctl*(2) states of the virtual console to those modes saved in the file **/etc/ioctl.syscon**. This file is written by *init* whenever the single-user state is entered.

When a run level change request is made *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target run level. *init* waits 5 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

The shell running on each terminal will terminate when the user types an end-of-file or hangs up. When *init* receives a signal telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** and **/etc/wtmp** if it exists [see *who*(1)]. A history of the processes spawned is kept in **/etc/wtmp.**

If *init* receives a *powerfail* signal (**SIGPWR**) it scans **/etc/inittab** for special entries of the type *powerfail* and *powerwait*. These entries are invoked (if the run levels permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions during the power-down of the operating system. Note that in the single-user states, **S** and **s**, only *powerfail* and *powerwait* entries are executed.

telinit

telinit, which is linked to **/etc/init**, is used to direct the actions of *init*. It takes a one-character argument and signals *init* to take the appropriate action.

FILES

/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/console
/dev/contty

SEE ALSO

getty(1M), login(1), sh(1), shutdown(1M), stty(1), who(1), gettydefs(4), inittab(4), utmp(4), termio(7).
kill(2) in the *Programmer's Reference Manual.*

DIAGNOSTICS

If *init* finds that it is respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string in the entry, and generate an error message on the system console. It will then refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user-spawned *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in **/etc/inittab**.

When attempting to boot the system, failure of *init* to prompt for a new run level may be because the virtual system console is linked to a device other than the physical system console.

WARNINGS

init and *telinit* can be run only by someone who is super-user.

The **S** or **s** state must not be used indiscriminately in the **/etc/inittab** file. A good rule to follow when modifying this file is to avoid adding this state to any line other than the **initdefault**.

The change to **/etc/gettydefs** described in the **WARNINGS** section of the *gettydefs*(4) manual page will permit terminals to pass 8 bits to the system as long as the system is in multi-user state (run level greater than 1). When the system changes to single-user state, the *getty* is killed and the terminal attributes are lost. To permit a terminal to pass 8 bits to the system in single-user state, after you are in single-user state, type:

```
stty -istrip cs8
```

The **/etc/TIMEZONE** file must exist.

NAME
installpkg – install package

SYNOPSIS
installpkg

DESCRIPTION
The *installpkg* command is used to install a UNIX system software package on the AT&T 386 UNIX System. It will install software that conforms to the PC 6300 PLUS Installation specificiation and the *sysadm* installation specifications.

You will have to be *root* to install certain packages successfully.

You will be prompted to insert the floppy disk that the installation package resides on. Everything else is automatic.

LIMITATIONS
You must envoke *installpkg* on the console.

SEE ALSO
displaypkg(1), removepkg(1).

NAME

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

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

The *ipcrm* command will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

-q *msqid*	removes the message queue identifier *msqid* 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) in the *Programmer's Reference Manual.*

NAME

ipcs – report interprocess communication facilities status

SYNOPSIS

ipcs [options]

DESCRIPTION

The *ipcs* command prints certain information about active interprocess 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 subject to these options:

-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, 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, process ID of last process to receive a message on message queues, 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.

KEY (all)
: The key used as an argument to *msgget, semget,* or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

MODE (all)
: The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:
The first two characters are:

R	if a process is waiting on a *msgrcv;*
S	if a process is waiting on a *msgsnd;*
D	if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
C	if the associated shared memory segment is to be cleared when the first attach is executed;
-	if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

r	if read permission is granted;
w	if write permission is granted;
a	if alter permission is granted;
-	if the indicated permission is *not* granted.

OWNER (all)
: The login name of the owner of the facility entry.

GROUP (all)
: The group name of the group of the owner of the facility entry.

CREATOR	(a,c) The login name of the creator of the facility entry.
CGROUP	(a,c) The group name of the group of the creator of the facility entry.
CBYTES	(a,o) The number of bytes in messages currently outstanding on the associated message queue.
QNUM	(a,o) The number of messages currently outstanding on the associated message queue.
QBYTES	(a,b) The maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID	(a,p) The process ID of the last process to send a message to the associated queue.
LRPID	(a,p) The process ID of the last process to receive a message from the associated queue.
STIME	(a,t) The time the last message was sent to the associated queue.
RTIME	(a,t) The time the last message was received from the associated queue.
CTIME	(a,t) The time when the associated entry was created or changed.
NATTCH	(a,o) The number of processes attached to the associated shared memory segment.
SEGSZ	(a,b) The size of the associated shared memory segment.
CPID	(a,p) The process ID of the creator of the shared memory entry.
LPID	(a,p) The process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t) The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t) The time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b) The number of semaphores in the set associated with the semaphore entry.

OTIME	(a,t) The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	system name list
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

WARNING

The user's real UID and effective UID must be the same.

The user's real GID and effective GID must be the same.

SEE ALSO

msgop(2), semop(2), shmop(2) in the *Programmer's Reference Manual.*

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

NAME

ismpx – return windowing terminal state

SYNOPSIS

ismpx [-s]

DESCRIPTION

The *ismpx* command reports whether its standard input is connected to a multiplexed *xt*(7) channel; i.e., whether it's running under *layers*(1) or not. It is useful for shell scripts that download programs to a windowing terminal or depend on screen size.

The *ismpx* command prints **yes** and returns **0** if invoked under *layers*(1), and prints **no** and returns **1** otherwise.

-s Do not print anything; just return the proper exit status.

EXIT STATUS

Returns **0** if invoked under *layers*(1), **1** if not.

SEE ALSO

jwin(1), layers(1), xt(7).

EXAMPLE

```
if ismpx -s
then
        jwin
fi
```

NAME

join – relational data base operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

The *join* command 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 [see *sort*(1)].

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*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively. The following options are recognized:

-a*n*	In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
-e *s*	Replace empty output fields by string *s*.
-j*n m*	Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with **1**.
-o *list*	Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.
-t*c*	Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
join –j1 4 –j2 3 –o 1.1 2.1 1.6 –t: /etc/passwd /etc/group
```

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

BUGS

With default field separation, the collating sequence is that of **sort –b**; with **–t**, the sequence is that of a plain sort.

The conventions of *join, sort, comm, uniq,* and *awk*(1) are wildly incongruous.

File names that are numeric may cause conflict when the **-o** option is used right before listing file names.

NAME

jterm – reset layer of windowing terminal

SYNOPSIS

jterm

DESCRIPTION

The *jterm* command is used to reset a layer of a windowing terminal after downloading a terminal program that changes the terminal attributes of the layer. It is useful only under *layers*(1). In practice, it is most commonly used to restart the default terminal emulator after using an alternate one provided with a terminal-specific application package. For example, on the AT&T TELETYPE 5620 DMD terminal, after executing the *hp2621*(1) command in a layer, issuing the *jterm* command will restart the default terminal emulator in that layer.

EXIT STATUS

Returns **0** upon successful completion, **1** otherwise.

NOTE

The layer that is reset is the one attached to standard error; that is, the window you are in when you type the *jterm* command.

SEE ALSO

layers(1).

NAME

jwin – print size of layer

SYNOPSIS

jwin

DESCRIPTION

The *jwin* command runs only under *layers*(1) and is used to determine the size of the layer associated with the current process. It prints the width and the height of the layer in bytes (number of characters across and number of lines, respectively). For bit-mapped terminals only, it also prints the width and height of the layer in bits.

EXIT STATUS

Returns **0** on successful completion, **1** otherwise.

DIAGNOSTICS

If *layers*(1) has not been invoked, an error message is printed:

```
jwin: not mpx
```

NOTE

The layer whose size is printed is the one attached to standard input; that is, the window you are in when you type the *jwin* command.

SEE ALSO

layers(1).

EXAMPLE

```
jwin
bytes:  86 25
bits:   780 406
```

NAME

kill – terminate a process

SYNOPSIS

kill [-signo] PID ...

DESCRIPTION

The *kill* command 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) in the *Programmer's Reference Manual.*

NAME

killall – kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

The *killall* command is used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure.

The *killall* command terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

The *killall* command sends *signal* [see *kill*(1)] to all processes not belonging to the above group of exclusions. If no *signal* is specified, a default of **9** is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), kill(1), ps(1), shutdown(1M).
signal(2) in the *Programmer's Reference Manual.*

WARNINGS

The *killall* command can be run only by the super-user.

NAME

labelit – provide labels for file systems

SYNOPSIS

/etc/labelit special [fsname volume [**-n**]]

DESCRIPTION

The *labelit* command can be used to provide labels for unmounted disk file systems or file systems being copied to tape. The **-n** option provides for initial labeling only. (This destroys previous contents.)

With the optional arguments omitted, *labelit* prints current label values.

The *special* name should be the physical disk section (e.g., **/dev/dsk/0s3**). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (e.g., **root**, **u1**, etc.) of the file system.

Volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette, or tape.

For file systems on disk, *fsname* and *volume* are recorded in the super block.

SEE ALSO

sh(1).
fs(4) in the *Programmer's Reference Manual.*

NAME

layers – layer multiplexer for windowing terminals

SYNOPSIS

layers [**-s**] [**-t**] [**-d**] [**-p**] [**-f** file] [layersys-prgm]

DESCRIPTION

The *layers* command manages asynchronous windows [see *layers*(5)] on a windowing terminal. Upon invocation, *layers* finds an unused *xt*(7) channel group and associates it with the terminal line on its standard output. It then waits for commands from the terminal.

Command-line options:

-s Reports protocol statistics on standard error at the end of the session after you exit from *layers*. The statistics may be printed during a session by invoking the program *xts*(1M).

-t Turns on *xt*(7) driver packet tracing, and produces a trace dump on standard error at the end of the session after you exit from *layers*. The trace dump may be printed during a session by invoking the program *xtt*(1M).

-d If a firmware patch has been downloaded, prints out the sizes of the text, data, and bss portions of the firmware patch on standard error.

-p If a firmware patch has been downloaded, prints the down-loading protocol statistics and a trace on standard error.

-f *file* Starts *layers* with an initial configuration specified by *file*. Each line of the file represents a layer to be created, and has the following format:

```
origin_x    origin_y    corner_x    corner_y
command_list
```

The coordinates specify the size and position of the layer on the screen in the terminal's coordinate system. If all four are *0*, the user must define the layer interactively. *command_list*, a list of one or more commands, must be provided. It is executed in the newlayer using the user's shell (by executing: **$SHELL -i -c** "*command_list*"). This means that the last command should invoke a shell, such as */bin/sh*. (If the last command is not a shell, then, when the last command has completed, the layer will not be functional.)

layersys-prgm
A file containing a firmware patch that the *layers* command downloads to the terminal before layers are created and *command_list* is executed.

Each layer is in most ways functionally identical to a separate terminal. Characters typed on the keyboard are sent to the standard input of the UNIX system process attached to the current layer (called the host process), and characters written on the standard output by the host process appear in that layer. When a layer is created, a separate shell is established and bound to the layer. If the environment variable **SHELL** is set, the user will get that shell: otherwise, */bin/sh* will be used. In order to enable communications with other users via *write*(1), *layers* invokes the command *relogin*(1M) when the first layer is created. *relogin*(1M) will reassign that layer as the user's logged-in terminal. An alternative layer can be designated by using *relogin*(1M) directly. *layers* will restore the original assignment on termination.

Layers are created, deleted, reshaped, and otherwise manipulated in a terminal-dependent manner. For instance, the AT&T TELETYPE 5620 DMD terminal provides a mouse-activated pop-up menu of layer operations. The method of ending a *layers* session is also defined by the terminal.

EXAMPLE

```
layers -f startup
```

where **startup** contains

```
8 8 700 200 date ; pwd ; exec $SHELL
8 300 780 850 exec $SHELL
```

NOTES

The *xt*(7) driver supports an alternate data transmission scheme known as ENCODING MODE. This mode makes *layers* operation possible even over data links which intercept control characters or do not transmit 8-bit characters. ENCODING MODE is selected either by setting a configuration option on your windowing terminal or by setting the environment variable **DMDLOAD** to the value *hex* before running *layers*:

```
    export DMDLOAD; DMDLOAD=hex
```

If, after executing **layers -f** *file*, the terminal does not respond in one or more of the layers, often the last command in the *command-list* for that layer did not invoke a shell.

WARNING

When invoking *layers* with the **-s**, **-t**, **-d**, or **-p** options, it is best to redirect standard error to another file to save the statistics and tracing output (e.g., **layers -s 2>stats**); otherwise all or some of the output may be lost.

FILES

/dev/xt??[0-7]
/usr/lib/layersys/lsys.8;7;3
/usr/lib/layersys/lsys.8;?;?

SEE ALSO

relogin(1M), sh(1), write(1), wtinit(1M), xts(1M), xtt(1M), xt(7).

libwindows(3X), layers(5) in the *Programmer's Reference Manual.*

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

The *line* command 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) in the *Programmer's Reference Manual.*

NAME

link, unlink – link and unlink files and directories

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

The *link* command is used to create a file name that points to another file. Linked files and directories can be removed by the *unlink* command; however, it is strongly recommended that the *rm*(1) and *rmdir*(1) commands be used instead of the *unlink* command.

The only difference between *ln*(1) and *link/unlink* is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the *link*(2) and *unlink*(2) system calls.

SEE ALSO

rm(1).
link(2), unlink(2) in the *Programmer's Reference Manual.*

WARNINGS

These commands can be run only by the super-user.

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 make any mistake in the login procedure you will receive the message:

Login incorrect

and a new login prompt will appear. If you make five incorrect login attempts, all five will be logged in LOGINLOG (defined to be **/usr/adm/loginlog**) and the line will be dropped.

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 path name (i.e., **-sh**). If this field in the password file is empty, then the default command interpreter, **/bin/sh** is used. If this field is "*", then the named directory becomes the root directory, the starting point for path searches for path names beginning with a **/**. At that point *login* is re-executed at the new level which must have its own root structure, including **/etc/login** and **/etc/passwd**.

The basic *environment* 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

L*n*=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, 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 are not 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

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/*your-name*	mailbox for user *your-name*
/usr/adm/loginlog	record of failed login attempts
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile
.profile	user's login profile

SEE ALSO

mail(1), newgrp(1M), sh(1), su(1M).

loginlog(4), passwd(4), profile(4), environ(5) in the *Programmer's Reference Manual.*

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 system 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.

NAME

logname – get login name

SYNOPSIS

logname

DESCRIPTION

The *logname* command returns the contents of the environment variable **$LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1).
logname(3X), environ(5) in the *Programmer's Reference Manual.*

NAME

lp, cancel – send/cancel requests to an LP line printer

SYNOPSIS

lp [**-c**] [**-d**dest] [**-m**] [**-n**number] [**-o**option] [**-s**] [**-t**title] [**-w**] files
cancel [ids] [printers]

DESCRIPTION

The *lp* command 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.

The *lp* command 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.

The *cancel* command 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

accept(1M), enable(1), lpadmin(1M), lpsched(1M), lpstat(1), mail(1).

NAME

lpadmin – configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin -pprinter [options]
/usr/lib/lpadmin -xdest
/usr/lib/lpadmin -d[dest]

DESCRIPTION

The *lpadmin* command configures line printer (LP) spooling systems to describe printers, classes, and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs, and to change the system default destination. *lpadmin* may not be used when the LP scheduler, *lpsched*(1M), is running, except where noted below.

Exactly one of the **-p**, **-d**, or **-x** options must be present for every legal invocation of *lpadmin*.

-p*printer*	names a *printer* to which all of the *options* below refer. If *printer* does not exist, then it will be created.
-x*dest*	removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other *options* are allowed with **-x**.
-d[*dest*]	makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(1M) is running. No other *options* are allowed with **-d**.

The following *options* are only useful with **-p** and may appear in any order. For ease of discussion, the printer will be referred to as *P* below.

-c*class*	inserts printer *P* into the specified *class*. *Class* will be created if it does not already exist.
-e*printer*	copies an existing *printer's* interface program to be the new interface program for *P*.
-h	indicates that the device associated with *P* is hard-wired. This *option* is assumed when adding a new printer unless the **-l** *option* is supplied.
-i*interface*	establishes a new interface program for *P*. *Interface* is the path name of the new program.
-l	indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*, disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.
-m*model*	selects a model interface program for *P*. *Model* is one of the model interface names supplied with the LP Spooling Utilities (see *Models* below).
-r*class*	removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* will be removed.

-v*device* associates a new *device* with printer *P*. *Device* is the pathname of a file that is writable by *lp*. Note that the same *device* can be associated with more than one *printer*. If only the **-p** and **-v** *options* are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions

When creating a new printer, the **-v** option and one of the **-e**, **-i**, or **-m** options must be supplied. Only one of the **-e**, **-i**, or **-m** options may be supplied. The **-h** and **-l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters **A**-**Z**, **a**-**z**, **0**-**9** and _ (underscore).

Models

Model printer interface programs are supplied with the LP Spooling Utilities. They are shell procedures which interface between *lpsched* and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **-m**. Copies of model interface programs may also be modified and then associated with printers using *lpadmin* **-i**. The following describes the *models* which may be given on the *lp* command line. The **-o** keyletter is used with the *lp* cpmmand to access options in the model files.

ATT455	Letter quality printer using XON/XOFF protocol at 9600 baud.
ATT473	Dot matrix draft quality printer using XON/XOFF protocol at 9600 baud.

EXAMPLES

1. For a ATT473 printer named cl8, it will use the DQP-10 model interface after the command:

```
/usr/lib/lpadmin -pcl8 -matt473
```

2. An ATT455 printer called pr1 can be added to the lp configuration with the command:

```
/usr/lib/lpadmin -ppr1 -v/dev/contty -matt455
```

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1).

NAME

lpsched, lpshut, lpmove – start/stop the LP scheduler and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2

DESCRIPTION

lpsched schedules requests taken by *lp*(1) for printing on line printers (LP's).

Lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again.

Lpmove moves requests that were queued by *lp*(1) between LP destinations. This command may be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination, *dest*. *Requests* are request ids as returned by *lp*(1). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp (1)* will reject requests for *dest1*.

Note that *lpmove* never checks the acceptance status [see *accept*(1M)] for the new destination when moving requests.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpstat(1).

NAME

lpstat – print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

The *lpstat* command prints information about the current status of the LP spooling 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 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 path names 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 directory

SYNOPSIS

ls [-RadCxmlnogrtucpFbqisf] [names]

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the **-C** and **-x** options enable multi-column formats, and the **-m** option enables stream output format. In order to determine output formats for the **-C**, **-x**, and **-m** options, *ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *terminfo*(4) data base is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

The *ls* command has the following options:

-R Recursively list subdirectories encountered.

-a List all entries, including those that begin with a dot (.), which are normally not listed.

-d If an argument is a directory, list only its name (not its contents); often used with **-l** to get the status of a directory.

-C Multi-column output with entries sorted down the columns.

-x Multi-column output with entries sorted across rather than down the page.

-m Stream output format; files are listed across the page, separated by commas.

-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 instead contain the major and minor device numbers rather than a size.

-n The same as **-l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

-o The same as **-l**, except that the group is not printed.

-g The same as **-l**, except that the owner is not printed.

-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

-t Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See **-n** and **-c**.)

-u	Use time of last access instead of last modification for sorting (with the **-t** option) or printing (with the **-l** option).
-c	Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**-t**) or printing (**-l**).
-p	Put a slash (/) after each file name if that file is a directory.
-F	Put a slash (/) after each file name if that file is a directory and put an asterisk (*) after each file name if that file is executable.
-b	Force printing of non-printable characters to be in the octal **ddd** notation.
-q	Force printing of non-printable characters in file names as the character question mark (?).
-i	For each file, print the i-number in the first column of the report.
-s	Give size in blocks, including indirect blocks, for each entry.
-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.

The mode printed under the **-l** option consists of ten characters. The first character may be one of the following:

d	the entry is a directory;
b	the entry is a block special file;
c	the entry is a character special file;
p	the entry is a fifo (a.k.a. "named pipe") special file;
-	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.

ls -l (the **l**ong list) prints its output as follows:

```
-rwxrwxrwx  1 smith  dev    10876  May 16 9:42 part2
```

This horizontal configuration provides a good deal of information. Reading from right to left, you see that the current directory holds one file, named "part2." Next, the last time that file's contents were modified was 9:42 A.M. on May 16. The file is moderately sized, containing 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group "dev" (perhaps indicating "development"), and his or her login name is "smith." The number, in this case "1," indicates the number of links to file "part2." Finally, the row of dash and letters tell you that user, group, and others have permissions to read, write, execute "part2."

The execute (**x**) symbol here occupies the third position of the three-character sequence. A **-** in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

r	the file is readable
w	the file is writable
x	the file is executable
-	the indicated permission is *not* granted
l	mandatory locking will occur during access (the set-group-ID bit is on and the group execution bit is off)
s	the set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on
S	undefined bit-state (the set-user-ID bit is on and the user execution bit is off)
t	the 1000 (octal) bit, or sticky bit, is on [see *chmod*(1)], and execution is on
T	the 1000 bit is turned on, and execution is off (undefined bit-state)

For user and group permissions, the third position is sometimes occupied by a character other than **x** or **-**. **s** also may occupy this position, referring to the state of the **s**et-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user stated at "login."

In the case of the sequence of group permissions, **l** may occupy the third position. **l** refers to mandatory file and record **l**ocking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by **t** or **T**. These refer to the state of the sticky bit and execution permissions.

EXAMPLES

An example of a file's permissions is:

```
-rwxr--r--
```

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

Another example of a file's permissions is:

```
-rwsr-xr-x
```

This describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

Another example of a file's permissions is:

```
-rw-rwl---
```

This describes a file that is readable and writable only by the user and the group and can be locked during access.

An example of a command line:

ls -a

This command will print the names of all files in the current directory, including those that begin with a dot (.), which normally do not print.

Another example of a command line:

ls -aisn

This command will provide you with quite a bit of information including **a**ll files, including non-printing ones (**a**), the **i**-number—the memory address of the i-node associated with the file—printed in the left-hand column (**i**); the **s**ize (in blocks) of the files, printed in the column to the right of the i-numbers (**s**); finally, the report is displayed in the **n**umeric version of the long list, printing the **UID** (instead of user name) and **GID** (instead of group name) numbers associated with the files.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	user IDs for **ls -l** and **ls -o**
/etc/group	group IDs for **ls -l** and **ls -g**
/usr/lib/terminfo/?/*	terminal information database

SEE ALSO

chmod(1), find(1).

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls -l** command leads you to believe. For more information see the "Mapping Remote Users" section of Chapter 10 of the *System Administrator's Guide*.

BUGS

Unprintable characters in file names may confuse the columnar output options.

NAME

machid: i386 – get processor type truth value

SYNOPSIS

i386

DESCRIPTION

The following commands will return a true value (exit code of 0).

The commands that do not apply will return a false (non-zero) value. These commands are often used within makefiles [see *make*(1)] and shell procedures [see *sh*(1)] to increase portability.

SEE ALSO

sh(1), test(1), true(1).
make(1) in the *Programmer's Reference Manual*.

NAME

mail, rmail – send mail to users or read mail

SYNOPSIS

Sending mail:

mail [**-wt**] persons

rmail [**-wt**] persons

Reading mail:

mail [**-ehpqr**] [**-f** file] [**-F** persons]

DESCRIPTION

Sending mail:

The command-line arguments that follow affect SENDING mail:

-w causes a letter to be sent to a remote user without waiting for the completion of the remote transfer program.

-t causes a **To:** line to be added to the letter, showing the intended recipients.

A *person* is usually a user name recognized by *login*(1). When *persons* are named, *mail* assumes a message is being sent (except in the case of the –F option). It reads from the standard input up to an end-of-file (control-d), or until it reads a line consisting of just a period. When either of those signals is received, *mail* adds the *letter* to the *mailfile* for each *person*. A *letter* is a *message* preceded by a *postmark*. The message is preceded by the sender's name and a *postmark*. A *postmark* consists of one or more 'From' lines followed by a blank line.

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If *mail* is interrupted during input, the file **dead.letter** is saved to allow editing and resending. **dead.letter** is recreated every time it is needed, erasing any previous contents.

rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

If the local system has the Basic Networking Utilities installed, mail may be sent to a recipient on a remote system. Prefix *person* by the system name and exclamation point. A series of system names separated by exclamation points can be used to direct a letter through an extended network.

Reading Mail:

The command-line arguments that follow affect READING 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.

-h causes a window of headers to be displayed rather than the latest message. The display is followed by the '**?**' prompt.

-p causes all messages to be printed without prompting for disposition.

-q causes *mail* to terminate after interrupts. Normally an interrupt causes only 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*.

-F*persons*

entered into an empty *mailbox*, causes all incoming mail to be forwarded to *persons*.

mail, unless otherwise influenced by command-line arguments, prints a user's mail messages in last-in, first-out order. For each message, the user is prompted with a **?**, and a line is read from the standard input. The following commands are available to determine the disposition of the message:

<new-line>, +, or **n**	Go on to next message.
d, or **dp**	Delete message and go on to next message.
d #	Delete message number #. Do not go on to next message.
dq	Delete message and quit *mail*.
h	Display a window of headers around current message.
h #	Display header of message number #.
h a	Display headers of ALL messages in the user's *mailfile*.
h d	Display headers of messages scheduled for deletion.
p	Print current message again.
-	Print previous message.
a	Print message that arrived during the *mail* session.
#	Print message number #.
r [*users*]	Reply to the sender and other *user(s)*, then delete the message.
s [*files*]	Save message in the named *files* (**mbox** is default).
y	Same as save.
u [**#**]	Undelete message number # (default is last read).
w [*files*]	Save message, without its top-most header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons*.
q, or **ctl-d**	Put undeleted mail back in the *mailfile* and quit *mail*.
x	Put all mail back in the *mailfile* unchanged and exit *mail*.
!*command*	Escape to the shell to do *command*.
?	Print a command summary.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

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*. A "Forwarded by..." message is added to the header. This is especially useful in a multi-machine environment to forward all of a person's mail to a single machine, and to keep the recipient informed if the mail has been forwarded. Installation and removal of forwarding is done with the **-F** option.

To forward all of one's mail to systema!user enter:

```
mail –Fsystema!user
```

To forward to more than one user enter:

```
mail –F"user1,systema!user2,systema!systemb!user3"
```

Note that when more than one user is specified, the whole list should be enclosed in double quotes so that it may all be interpreted as the operand of the **-F** option. The list can be up to 1024 bytes; either commas or white space can be used to separate users.

The following list of characters are prohibited from appearing anywhere in the mail **-F** argument list:

```
; & | ^ < > ' ' " ? * [ ] ( ) { } $ # \ ~
```

To remove forwarding enter:

```
mail –F ""
```

The pair of double quotes is mandatory to set a NULL argument for the **-F** option.

In order for forwarding to work properly the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*user*	incoming mail for *user*; i.e., the *mailfile*
$HOME/mbox	saved mail
$MAIL	variable containing path name of *mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), mailx(1), write(1).

User's/System Administrator's Guide.

WARNING

The "Forward to person" feature may result in a loop, if *sys1!userb* forwards to *sys2!userb* and *sys2!userb* forwards to *sys1!userb*. The symptom is a message saying "unbounded...saved mail in dead.letter."

BUGS

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

mailx – interactive message processing system

SYNOPSIS

mailx [*options*] [*name*...]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing, and other modification of the message as it is entered.

Many of the remote features of *mailx* will only work if the Basic Networking Utilities are installed on your system.

Incoming mail is stored in a standard file for each user, called the *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's HOME directory [see "MBOX" (ENVIRONMENT VARIABLES) for a description of this file]. Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the **-f** option of the *mailx* command. Messages in the secondary file can then be read or otherwise processed using the same COMMANDS as in the primary *mailbox*. This gives rise within these pages to the notion of a current *mailbox*.

On the command line, *options* start with a dash (–) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the *mailbox*. Command line options are:

-e	Test for presence of mail. *mailx* prints nothing and exits with a successful return code if there is mail to read.
-f [*filename*]	Read messages from *filename* instead of *mailbox*. If no *filename* is specified, the *mbox* is used.
-F	Record the message in a file named after the first recipient. Overrides the "record" variable, if set (see ENVIRONMENT VARIABLES).
-h *number*	The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops. (See **addsopt** under ENVIRONMENT VARIABLES)
-H	Print header summary only.
-i	Ignore interrupts. See also "ignore" (ENVIRONMENT VARIABLES).
-n	Do not initialize from the system default *mailx.rc* file.

-N	Do not print initial header summary.
-r *address*	Pass *address* to network delivery software. All tilde commands are disabled. (See **addsopt** under ENVIRONMENT VARIABLES)
-s *subject*	Set the Subject header field to *subject*.
-u *user*	Read *user's mailbox*. This is only effective if *user's mailbox* is not read protected.
-U	Convert *uucp* style addresses to internet standards. Overrides the "conv" environment variable. (See **addsopt** under ENVIRONMENT VARIABLES)

When reading mail, *mailx* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see COMMANDS below). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. (A "subject" longer than 1024 characters will cause *mailx* to dump core) As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the **se**t and **uns**et commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to to undeliverable, an attempt is made to return it to the sender's *mailbox*. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp*(1) for recording outgoing mail on paper. Alias groups are set by the **a**lias command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, **p**rint is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers separated by spaces, which may include:

n	Message number **n**.
.	The current message.
^	The first undeleted message.
$	The last message.
*****	All messages.
n-m	An inclusive range of message numbers.
user	All messages from **user**.
/string	All messages with **string** in the subject line (case ignored).
:*c*	All messages of type *c*, where *c* is one of:

d	deleted messages
n	new messages
o	old messages
r	read messages
u	unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions [see *sh*(1)]. Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* tries to execute commands from the optional system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common use of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, **C**opy, **e**dit, **fo**llowup, **F**ollowup, **ho**ld, **m**ail, **pre**serve, **r**eply, **R**eply, **sh**ell, and **v**isual. An error in the start-up file causes the remaining lines in the file to be ignored. The **.mailrc** file is optional and must be constructed locally.

COMMANDS

The following is a complete list of *mailx* commands:

!*shell-command*

Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).

*comment*

Null command (comment). This may be useful in *.mailrc* files.

=

Print the current message number.

?

Prints a summary of commands.

alias *alias name ...*
group *alias name ...*

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

alternates *name* ...
: Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **alt**ernates prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).

cd [*directory*]
chdir [*directory*]
: Change directory. If *directory* is not specified, $HOME is used.

copy [*filename*]
copy [*msglist*] *filename*
: Copy messages to the file without marking the messages as saved. Otherwise equivalent to the **s**ave command.

Copy [*msglist*]
: Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the **S**ave command.

delete [*msglist*]
: Delete messages from the *mailbox*. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).

discard [*header-field* ...]
ignore [*header-field* ...]
: Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The **P**rint and **T**ype commands override this command.

dp [*msglist*]
dt [*msglist*]
: Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a **d**elete command followed by a **p**rint command.

echo *string* ...
: Echo the given strings [like *echo*(1)].

edit [*msglist*]
: Edit the given messages. The messages are placed in a temporary file and the "EDITOR" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is *ed*(1).

exit
xit

Exit from *mailx,* without changing the *mailbox.* No messages are saved in the *mbox* (see also **q**uit).

file [*filename*]
folder [*filename*]

Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

%	the current *mailbox.*
%**user**	the *mailbox* for **user**.
#	the previous file.
&	the current *mbox.*

Default file is the current *mailbox.*

folders

Print the names of the files in the directory set by the "folder" variable (see ENVIRONMENT VARIABLES).

followup [*message*]

Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the **F**ollowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

Followup [*msglist*]

Respond to the first message in the *msglist,* sending the message to the author of each message in the *msglist.* The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the **fo**llowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

from [*msglist*]

Prints the header summary for the specified messages.

group *alias name ...*
alias *alias name ...*

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

headers [*message*]

Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the **z** command.

help
 Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]
 Holds the specified messages in the *mailbox.*

if *s* | *r*
s
else
s
endif
 Conditional execution, where *s* will execute following s, up to an **el**se or **en**dif, if the program is in *send* mode, and *r* causes the s to be executed only in *receive* mode. Useful in the *.mailrc* file.

ignore *header-field ...*
discard *header-field ...*
 Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The **P**rint and **T**ype commands override this command.

list
 Prints all commands available. No explanation is given.

mail *name ...*
 Mail a message to the specified users.

Mail *name*
 Mail a message to the specified user and record a copy of it in a file named after that user.

mbox [*msglist*]
 Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a description of this file. See also the **ex**it and **q**uit commands.

next [*message*]
 Go to next message matching *message.* A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglist*s above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]
Pipe the message through the given *shell-command.* The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [*msglist*]
hold [*msglist*]
Preserve the specified messages in the *mailbox.*

Print [*msglist*]
Type [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command.

print [*msglist*]
type [*msglist*]
Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

quit
Exit from *mailx,* storing messages that were read in *mbox* and unread messages in the *mailbox.* Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]
Send a response to the author of each message in the *msglist.* The subject line is taken from the first message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

reply [*message*]
respond [*message*]
Reply to the specified message, including all other recipients of the message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

Save [*msglist*]
Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the **C**opy, **fo**llowup, and **F**ollowup commands and "outfolder" (ENVIRONMENT VARIABLES).

save [*filename*]
save [*msglist*] *filename*

Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the **exit** and **quit** commands).

set
set *name*
set *name=string*
set *name=number*

Define a variable called *name*. The variable may be given a null, string, or numeric value. **Set** by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the *mailx* variables.

shell

Invoke an interactive shell [see also "SHELL" (ENVIRONMENT VARIABLES)].

size [*msglist*]

Print the size in characters of the specified messages.

source *filename*

Read commands from the given file and return to command mode.

top [*msglist*]

Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

touch [*msglist*]

Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox*, or the file specified in the MBOX environment variable, upon normal termination. See **ex**it and **q**uit.

Type [*msglist*]
Print [*msglist*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command.

type [*msglist*]
print [*msglist*]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

undelete [*msglist*]
: Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

unset *name* ...
: Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version
: Prints the current version and release date.

visual [*msglist*]
: Edit the given messages with a screen editor. The messages are placed in a temporary file and the "VISUAL" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

write [*msglist*] *filename*
: Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the **s**ave command.

xit
exit
: Exit from *mailx,* without changing the *mailbox.* No messages are saved in the *mbox* (see also **q**uit).

z[+ ! –]
: Scroll the header display forward or backward one full screen. The number of headers displayed is set by the "screen" variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from *input mode,* by beginning a line with the tilde escape character (~). See "escape" (ENVIRONMENT VARIABLES) for changing this special character.

~! *shell-command*
: Escape to the shell.

~.
: Simulate end of file (terminate message input).

~:
~_
: Perform the command-level request. Valid only when sending a message while reading mail.

~?
Print a summary of tilde escapes.

~A
Insert the autograph string "Sign" into the message (see ENVIRONMENT VARIABLES).

~a
Insert the autograph string "sign" into the message (see ENVIRONMENT VARIABLES).

~b *name ...*
Add the *name*s to the blind carbon copy (Bcc) list.

~c *name ...*
Add the *name*s to the carbon copy (Cc) list.

~d
Read in the *dead.letter* file. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~e
Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT VARIABLES).

~f [*msglist*]
Forward the specified messages. The messages are inserted into the message without alteration.

~h
Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~i *string*
Insert the value of the named variable into the text of the message. For example, **~A** is equivalent to '**~i** Sign.' Environment variables set and exported in the shell are also accessible by ~i.

~m [*msglist*]
Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~p
Print the message being entered.

~q
Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead.letter*. See

"DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~r *filename*
~~< *filename*
~~< !*shell-command*

Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~s *string* ...

Set the subject line to *string*.

~t *name* ...

Add the given *name*s to the To list.

~v

Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

~w *filename*

Write the partial message onto the given file, without the header.

~x

Exit as with **~q** except the message is not saved in *dead.letter*.

~| *shell-command*

Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx*.

HOME=*directory*

The user's base of operations.

MAILRC=*filename*

The name of the start-up file. Default is $HOME/.mailrc.

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the **set** command at any time. The **uns**et command may be used to erase variables.

addsopt

Enabled by default. If */bin/mail* is not being used as the deliverer, **noaddsopt** should be specified. (See WARNINGS below)

allnet
All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet.** See also the **alt**ernates command and the "metoo" variable.

append
Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend.**

askcc
Prompt for the Cc list after message is entered. Default is **noaskcc.**

asksub
Prompt for subject if it is not specified on the command line with the **-s** option. Enabled by default.

autoprint
Enable automatic printing of messages after **d**elete and **u**ndelete commands. Default is **noautoprint.**

bang
Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). Default is **nobang.**

cmd=*shell-command*
Set the default command for the **pi**pe command. No default value.

conv=*conversion*
Convert uucp addresses to the specified address style. The only valid conversion now is *internet,* which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also "sendmail" and the **-U** command line option.

crt=*number*
Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable [*pg*(1) by default]. Disabled by default.

DEAD=*filename*
The name of the file in which to save partial letters in case of untimely interrupt. Default is $HOME/dead.letter.

debug
Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug.**

dot

Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR=*shell-command*

The command to run when the **e**dit or **~e** command is used. Default is *ed*(1).

escape=*c*

Substitute *c* for the ~ escape character. Takes effect with next message sent.

folder=*directory*

The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a *mailx* command line, "folder" must be an exported *sh* environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header

Enable printing of the header summary when entering *mailx*. Enabled by default.

hold

Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is **nohold**.

ignore

Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof

Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the **~.** command. Default is **noignoreeof**. See also "dot" above.

keep

When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave

Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*

The name of the file to save messages which have been read. The **x**it command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

metoo
: If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

LISTER=*shell-command*
: The command (and options) to use when listing the contents of the "folder" directory. The default is *ls*(1).

onehop
: When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder
: Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the path name is absolute. Default is **nooutfolder**. See "folder" above and the **S**ave, **C**opy, **fo**llowup, and **F**ollowup commands.

page
: Used with the **pi**pe command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER=*shell-command*
: The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg*(1).

prompt=*string*
: Set the *command mode* prompt to *string*. Default is "? ".

quiet
: Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet**.

record=*filename*
: Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save
: Enable saving of messages in *dead.letter* on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen=*number*
: Sets the number of lines in a full screen of headers for the **h**eaders command.

sendmail=*shell-command*
Alternate command for delivering messages. Default is */bin/rmail*(1).

sendwait
Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL=*shell-command*
The name of a preferred command interpreter. Default is *sh*(1).

showto
When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string*
The variable inserted into the text of a message when the **~a** (autograph) command is given. No default [see also **~i** (TILDE ESCAPES)].

Sign=*string*
The variable inserted into the text of a message when the **~A** command is given. No default [see also **~i** (TILDE ESCAPES)].

toplines=*number*
The number of lines of header to print with the **top** command. Default is 5.

VISUAL=*shell-command*
The name of a preferred screen editor. Default is *vi*(1).

FILES

$HOME/.mailrc	personal start-up file
$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	optional global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO
ls(1), mail(1), pg(1).

WARNINGS
The **-h**, **-r** and **-U** options can be used only if *mailx* is built with a delivery program other than */bin/mail*.

BUGS
Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

The full internet addressing is not fully supported by *mailx*. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by *mail*(1) (the standard mail delivery program).

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

The *makekey* command 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 uppercase and lowercase 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 characters 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.

The *makekey* command is intended for programs that perform encryption. Usually, its input and output will be pipes.

SEE ALSO

ed(1), crypt(1), vi(1).
passwd(4) in the *Programmer's Reference Manual.*

WARNING

This command is provided with the Security Administration Utilities, which is only available in the United States.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [**-n**] [**-y**]

DESCRIPTION

The *mesg* command with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. The *mesg* command with argument **y** reinstates permission. All by itself, *mesg* 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 directories

SYNOPSIS

mkdir [**-m** mode] [**-p**] dirname ...

DESCRIPTION

The *mkdir* command creates the named directories in mode 777 [possibly altered by *umask*(1)].

Standard entries in a directory (e.g., the files **.**, for the directory itself, and **..**, for its parent) are made automatically. *mkdir* cannot create these entries by name. Creation of a directory requires write permission in the parent directory.

The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

Two options apply to *mkdir*:

-m This option allows users to specify the mode to be used for new directories. Choices for modes can be found in *chmod*(1).

-p With this option, *mkdir* creates *dirname* by creating all the non-existing parent directories first.

EXAMPLE

To create the subdirectory structure **ltr/jd/jan**, type:

```
mkdir -p ltr/jd/jan
```

SEE ALSO

sh(1), rm(1), umask(1).
intro(2), mkdir(2) in the *Programmer's Reference Manual*.

DIAGNOSTICS

The *mkdir* command returns exit code 0 if all directories given in the command line were made successfully. Otherwise, it prints a diagnostic and returns non-zero. An error code is stored in *errno*.

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs special blocks[:i-nodes] [gap blocks/cyl]
/etc/mkfs special proto [gap blocks/cyl]

DESCRIPTION

The *mkfs* command constructs a file system by writing on the *special* file using the values found in the remaining arguments of the command line. The command waits 10 seconds before starting to construct the file system. During this 10-second pause, the command can be aborted by entering a delete (DEL).

If the second argument is a string of digits, the size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* (512-byte) disk blocks the file system will occupy. If the number of i-nodes is not given, the default is the number of *logical* (1024-byte) blocks divided by 4. *mkfs* builds a file system with a single empty directory on it. The boot program block (block zero) is left uninitialized.

If the second argument is the name of a file that can be opened, *mkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. A sample prototype specification follows (line numbers have been added to aid in the explanation):

```
1.      /stand/diskboot
2.      4872 110
3.      d--777 3 1
4.      usr    d--777 3 1
5.             sh     ---755 3 1 /bin/sh
6.             ken    d--755 6 1
7.                    $
8.             b0     b--644 3 1 0 0
9.             c0     c--644 3 1 0 0
10.            $
11.     $
```

Line 1 in the example is the name of a file to be copied onto block zero as the bootstrap program.

Line 2 specifies the number of *physical* (512-byte) blocks the file system is to occupy and the number of i-nodes in the file system.

Lines 3-9 tell *mkfs* about files and directories to be included in this file system.

Line 3 specifies the root directory.

lines 4-6 and 8-9 specifies other directories and files.

The $ on line 7 tells *mkfs* to end the branch of the file system it is on, and continue from the next higher directory. The $ on lines 10 and 11 end the process, since no additional specifications follow.

File specifications give the mode, the user ID, the group ID, and the initial contents of the file. Valid syntax for the contents field depends on the first character of the mode.

The mode for a file is specified by a 6-character string. The first character specifies the type of the file. The character range is **-bcd** to specify regular, block special, character special and directory files respectively. The second character of the mode is either **u** or **-** to specify set-user-id mode or not. The third is **g** or **-** for the set-group-id mode. The rest of the mode is a 3-digit octal number giving the owner, group, and other read, write, execute permissions [see *chmod*(1)].

Two decimal number tokens come after the mode; they specify the user and group IDs of the owner of the file.

If the file is a regular file, the next token of the specification may be a path name whence the contents and size are copied. If the file is a block or character special file, two decimal numbers follow which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. As noted above, the scan is terminated with the token **$**.

The final argument in both forms of the command specifies the rotational *gap* and the number of *blocks/cyl*. The gap size should always be 2. If the *gap* and *blocks/cyl* are not specified or are considered illegal values a default value of gap size 7 and 400 blocks/cyl is used.

FILES

/etc/vtoc/*

SEE ALSO

chmod(1).

dir(4), fs(4) in the *Programmer's Reference Manual.*

BUGS

With a prototype file, it is not possible to copy in a file larger than 64K bytes, nor is there a way to specify links. The maximum number of i-nodes configurable is 65500.

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs special blocks[:i-nodes] [gap blocks/cyl] [–b blocksize]
/etc/mkfs special proto [gap blocks/cyl] [–b blocksize]

DESCRIPTION

mkfs constructs a file system by writing on the *special* file using the values found in the remaining arguments of the command line. The command waits 10 seconds before starting to construct the file system. During this 10-second pause the command can be aborted by entering a delete (DEL).

The *–b blocksize* option specifies the logical block size for the file system. The logical block size is the number of bytes read or written by the operating system in a single I/O operation. Valid values for *blocksize* are 512, 1024, and 2048. The default is 1024. A block size of 2048 may be chosen only if the 2K file system package is installed. If the **-b** option is used it must appear last on the command line.

If the second argument to *mkfs* is a string of digits, the size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* (512 byte) disk blocks the file system will occupy. If the number of i-nodes is not given, the default is approximately the number of *logical* blocks divided by 4. *mkfs* builds a file system with a single empty directory on it. The boot program block (block zero) is left uninitialized.

If the second argument is the name of a file that can be opened, *mkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. A sample prototype specification follows (line numbers have been added to aid in the explanation):

```
1.    /stand/ diskboot
2.    4872 110
3.    d--777 3 1
4.    usr  d--777 3 1
5.         sh   ---755 3 1 /bin/sh
6.         ken  d--755 6 1
7.              $
8.         b0   b--644 3 1 0 0
9.         c0   c--644 3 1 0 0
10.        $
11.   $
```

Line 1 in the example is the name of a file to be copied onto block zero as the bootstrap program.

Line 2 specifies the number of *physical* (512 byte) blocks the file system is to occupy and the number of i-nodes in the file system.

Lines 3-9 tell *mkfs* about files and directories to be included in this file system.

Line 3 specifies the root directory.

lines 4-6 and 8-9 specifies other directories and files.

The $ on line 7 tells *mkfs* to end the branch of the file system it is on, and continue from the next higher directory. The $ on lines 10 and 11 end the process, since no additional specifications follow.

File specifications give the mode, the user ID, the group ID, and the initial contents of the file. Valid syntax for the contents field depends on the first character of the mode.

The mode for a file is specified by a 6-character string. The first character specifies the type of the file. The character range is **-bcd** to specify regular, block special, character special and directory files respectively. The second character of the mode is either **u** or **-** to specify set-user-id mode or not. The third is **g** or **-** for the set-group-id mode. The rest of the mode is a 3 digit octal number giving the owner, group, and other read, write, execute permissions [see *chmod*(1)].

Two decimal number tokens come after the mode; they specify the user and group IDs of the owner of the file.

If the file is a regular file, the next token of the specification may be a path name whence the contents and size are copied. If the file is a block or character special file, two decimal numbers follow which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. As noted above, the scan is terminated with the token **$**.

The *gap blocks/cyl* argument in both forms of the command specifies the rotational gap and the number of blocks/cylinder.

FILES

/etc/vtoc/*

SEE ALSO

chmod(1).
dir(4), fs(4) in the *Programmer's Reference Manual.*

BUGS

With a prototype file, it is not possible to copy in a file larger than 64K bytes, nor is there a way to specify links. The maximum number of i-nodes configurable is 65500.

NAME

mknod – build special file

SYNOPSIS

/etc/mknod name **b** | **c** major minor
/etc/mknod name **p**

DESCRIPTION

The *mknod* command makes a directory entry and corresponding i-node for a special file.

The first argument is the *name* of the entry. The UNIX System convention is to keep such files in the /dev directory.

In the first case, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g., unit, drive, or line number). They may be either decimal or octal. The assignment of major device numbers is specific to each system. The information is contained in the system source file **conf.c**. You must be the super-user to use this form of the command.

The second case is the form of the *mknod* that is used to create FIFO's (a.k.a named pipes).

WARNING

If **mknod** is used to create a device in a remote directory (Remote File Sharing), the major and minor device numbers are interpreted by the server.

SEE ALSO

mknod(2) in the *Programmer's Reference Manual.*

NAME

mkpart – disk maintenance utility

SYNOPSIS

/etc/mkpart [**-f** filename] [**-p** partition] ... [**-P** partition] ... [**-b**] [**-B** filename] [**-A** sector] ... [**-V**] [**-v**] [**-i**] [**-x** file] [**-t** [vpa]] device

/etc/mkpart **-F** interleave raw_device

DESCRIPTION

This program allows the system administrator to display and modify the data structures that the disk driver uses to access disks. These structures describe the number, size, and type of the partitions, as well as the physical characteristics of the disk drive itself.

The user maintains a file of stanzas, each of which contains comments and parameters. The stanzas are of two varieties: those that describe disk partitions, and disk devices. Stanzas may refer to other stanzas of the same type so that common device or partition types may be customized. By default, the stanza file is named /etc/partitions. The required parameter, *device,* specifies the device stanza for the disk to be used.

The following options may be used with *mkpart*:

-f *filename*
: specifies the partition and device specification stanza file. If not present, /etc/partitions is assumed.

-p *partition*
: removes a partition from the vtoc on the specified device. The *partition* is a stanza that indicates the partition to be removed by its partition number parameter; no comparisons are made by attribute. NOTE: Alternate partitions cannot be removed.

-P *partition*
: adds a partition to the vtoc on the specified device. *partition* is a stanza which contains and/or refers to other stanzas that contain all of the necessary parameters for a vtoc partition.

-b causes only the boot program to be updated, unless other options are specified.

-B *filename*
: specifies a different boot program than the one given by the device stanza.

-F *interleave*
: causes the entire device to be hardware formatted. This process rewrites all the sector headers on each track of the disk, enabling subsequent access using normal reads and writes. *interleave* is the distance in physical sectors between each successive logical sector. Normal values are 1 for track-cache controllers, 3–4 for standard controllers. The device for this option must be a raw UNIX system device. The **-F** option precludes all other options, thus should be used alone.

-A *sector*
: marks the specified sector as bad and assigns it an alternate if possible. *sector* is a zero-based absolute sector number from the beginning of the drive. To compute a sector number given cylinder, head, and (0-based) sector in track, the formula is cylinder * (sectors-per-track * heads-per-cylinder) + head * (sectors-per-track) + sector.

-V causes a complete surface-analysis pass to be run. This first writes a data pattern (currently 0xe5 in every byte) to each sector of the disk, then reads each sector. Any errors are noted and the bad sectors found are added to the alternates table if possible.

-v causes a non-destructive surface-analysis pass to be run. This just reads every sector of the disk, noting bad sectors as above.

-i initializes the VTOC on the drive to default values, clearing any existing partition and bad-sector information which may have existed. This is the only way to remove an alternates partition and can be used to re-initialize a drive which may have obsolete or incorrect VTOC data on it.

-x *file* writes a complete *device* and partition stanza list for the specified *device* to file "filename". Note: The tags in the file are pseudo names used to identify the slice.

-t [*vpa*]
: creates a listing of the current vtoc. The sub-parameters specify pieces to be printed: a - alternate sectors, p - partitions, and v - vtoc and related structures.

The partitions file is composed of blank-line-separated stanzas. (Blank lines have only tabs and spaces between new-lines). Commentary consists of all text between a '#' and a new-line. Stanzas begin with an identifier followed by a ':', and are followed by a comma-separated list of parameters. Each parameter has a keyword followed by an '=' followed by a value. The value may be a number, another stanza's name, a double quoted string, or a parenthesis-surrounded, comma-separated list of numbers or ranges of numbers, as appropriate for the keyword. Numbers may be written as decimal, octal, or hexadecimal constants in the form familiar to C programmers.

Device specification stanzas may contain the following parameters:

usedevice = name
: causes the named stanza's parameters to be included in the device definition.

boot = string
: indicates that the string is the filename of a bootstrap program to install on the disk.

device = string
: gives the filename of the character special device for the disk.

heads = number
: specifies the number of tracks per cylinder on the device.

cyls = number
: is the number of cylinders on the disk.

sectors = number
: is the number of sectors per track.

bpsec = number
: is the number of bytes per sector.

dserial = string
: is an arbitrary string which is recorded in the volume label. (Multibus systems only)

vtocsec = number
: gives the sector number to use for the volume table of contents. NOTE: for AT386 systems, this number MUST be 17.

altsec = number
: is the sector to use for the alternate block table.

badsec = number-list
: lists the known bad sectors. These are appended to any specified in the command line or found during surface analysis.

Partition stanzas may have the following parameters:

usepart = name
: refers to another partition stanza.

partition = number
: gives this partition's entry number in the vtoc.

tag = tagname
: A partition tag specifies the purpose of the partition. The *tagnames* are reserved words which are presently used for identification purposes ONLY:
BACKUP means the entire disk.
ROOT is a root file system partition.
BOOT is a bootstrap partition.
SWAP is a partition that does not contain a file system.
USR is a partition that does contain a file system.
ALTS contains alternate sectors to which the driver re-maps bad sectors. Currently a maximum of 62 alternate sectors is supported.
OTHER is a partition that the UNIX system does not know how to handle, such as MS-DOS space.

perm = permname
: specifies a permission type for the partition. Permissions are not mutually exclusive.
RO indicates that the partition cannot be written upon. Normally, write access is granted (standard UNIX system file permissions notwithstanding).
NOMOUNT disallows the driver from mounting the file system that may be contained in the partition.

VALID indicates that the partition contains valid data. Any partition added with the **-A** flag will be marked VALID.

start = number
: is the starting sector number for the partition. NOTE: For AT386 systems, the root file system should start at the *second* track of the cylinder which is the beginning of the active UNIX system 'fdisk' partition. This allows space for the writing of the boot code.

size = number
: is the size, in sectors, of the partition.

When **mkpart** is run, it first attempts to read the volume label (for multibus systems) or the 'fdisk' table (for AT386 systems), the VTOC block, and the alternate sector table. If any of the structures is invalid or cannot be read, or if the **-i** flag is specified, the internal tables are initialized to default values for the device specified (taken from the device stanza in the partition file). If the **-F** flag is specified, the device is formatted. If either the **-V** or **-v** flag is specified, the appropriate surface analysis is performed. After these steps, partitions are deleted or added as required. Next, any bad sectors specified in the partition file, found during surface analysis, or specified in the command line with **-A** flags are merged into the alternate sectors table. Note that an alternates partition must exist for any bad-sector marking to occur, as bad sectors are assigned good alternates at this point. Finally, the boot program is written to track 0 of cylinder 0 (Multibus systems) or the cylinder where the active UNIX system 'fdisk' partition starts (AT386 systems). If **-b** was not the only parameter specified, the updated VTOC and alternates tables are written, and the disk driver is instructed to re-read the tables when the drive is opened the next time. When only **-t** is specified, only a listing is created and no updating occurs.

FILES

/etc/partitions /etc/boot /dev/rdsk/*s0

BUGS

Currently, very little consistency checking is done. No checks are made to ensure that the 'fdisk' partition table is consistent with the UNIX system partitions placed in the VTOC. If a DOS 'fdisk' partition is started at cylinder 0, DOS will happily overwrite the UNIX system VTOC.

NAME

mount, umount – mount and unmount file systems and remote resources

SYNOPSIS

/etc/mount [–r] [–f *fstyp*] *special directory*
/etc/mount [–r] [–c] –d *resource directory*
/etc/mount
/etc/umount *special*
/etc/umount –d *resource*

DESCRIPTION

File systems other than **root** (/) are considered *removable* in the sense that they can be either available to users or unavailable. *mount* announces to the system that *special,* a block special device or *resource,* a remote resource, is available to users from the mount point *directory. directory* must exist already; it becomes the name of the root of the newly mounted *special* or *resource.* A unique resource may be mounted only once (no multiple mounts).

mount, when entered with arguments, adds an entry to the table of mounted devices, **/etc/mnttab**. *umount* removes the entry. If invoked with no arguments, *mount* prints the entire mount table. If invoked with any of the following partial argument lists, *mount* will search **/etc/fstab** to fill in the missing arguments: *special,* **–d** *resource, directory,* or **–d** *directory.*

The following options are available:

-r	indicates that *special* or *resource* is to be mounted read-only. If *special* or *resource* is write-protected or read-only advertised, this flag must be used.
-d	indicates that *resource* is a remote resource that is to be mounted on *directory* or unmounted. To mount a remote resource, Remote File Sharing must be up and running and the resource must be advertised by a remote computer [see *rfstart*(1M) and *adv*(1M)]. If **–d** is not used, *special* must be a local block special device.
–c	indicates that remote reads and writes should not be cached in the local buffer pool. **–c** is used in conjunction with **–d**.
–f *fstyp*	indicates that *fstyp* is the file system type to be mounted. If this argument is omitted, it defaults to the **root** *fstyp.*
special	indicates the block special device that is to be mounted on *directory.*
resource	indicates the remote resource name that is to be mounted on a *directory.*
directory	indicates the directory mount point for *special* or *resource.* (The directory must already exist.)

umount announces to the system that the previously mounted *special* or *resource* is to be made unavailable. If invoked with *directory* or **–d** *directory, umount* will search **/etc/fstab** to fill in the missing argument(s).

mount can be used by any user to list mounted file systems and resources. Only a super-user can mount and unmount file systems.

FILES

/etc/mnttab	mount table
/etc/fstab	file system table

SEE ALSO

adv(1M), fuser(1M), nsquery(1M), rfstart(1M), rmntstat(1M), setmnt(1M), unadv(1M), fstab(4), mnttab(4).
mount(2), umount(2) in the *Programmer's Reference Manual.*
"Remote File Sharing" chapter, *System Administrator's Guide,* for guidelines on mounting remote resources.

DIAGNOSTICS

If the *mount*(2) system call fails, *mount* prints an appropriate diagnostic. *mount* issues a warning if the file system to be mounted is currently labeled under another name. A remote resource mount will fail if the resource is not available or if Remote File Sharing is not running or if it is advertised read-only and not mounted with **-r**.

umount fails if *special* or *resource* is not mounted or if it is busy. *special* or *resource* is busy if it contains an open file or some user's working directory. In such a case, you can use *fuser*(1M) to list and kill processes that are using *special* or *resource.*

WARNINGS

Physically removing a mounted file system diskette from the diskette drive before issuing the *umount* command damages the file system.

NAME

mountall, umountall – mount, unmount multiple file systems

SYNOPSIS

/etc/mountall [-] [**file-system-table**] ...
/etc/umountall [**-k**]

DESCRIPTION

These commands may be executed only by the super-user.

The **mountall** command is used to mount file systems according to a *file-system-table*. (**/etc/fstab** is the default file system table.) The special file name "–" reads from the standard input.

Before each file system is mounted, it is checked using *fsstat*(1M) to see if it appears mountable. If the file system does not appear mountable, it is checked, using *fsck*(1M), before the mount is attempted.

The **umountall** command causes all mounted file systems except **root** to be unmounted. The **-k** option sends a SIGKILL signal, via *fuser*(1M), to processes that have files open.

FILES

File-system-table format:

column 1	block special file name of file system
column 2	mount-point directory
column 3	"–r" if to be mounted read-only; "–d" if remote
column 4	(optional) file system type string
column 5+	ignored

White space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

A typical file-system-table might read:

```
/dev/dsk/0s1   /usr –r S51K
```

SEE ALSO

fsck(1M), fsstat(1M), fuser(1M), mount(1M).

signal(2), fstab(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

No messages are printed if the file systems are mountable and clean.

Error and warning messages come from *fsck*(1M), *fsstat*(1M), and *mount*(1M).

NAME

mvdir – move a directory

SYNOPSIS

/etc/mvdir dirname name

DESCRIPTION

The *mvdir* command moves directories within a file system. *Dirname* must be a directory. If *name* does not exist, it will be created as a directory. If *name* does exist, *dirname* will be created as *name/dirname. Dirname* and *name* may not be on the same path; that is, one may not be subordinate to the other. For example:

```
mvdir x/y x/z
```

is legal, but

```
mvdir x/y x/y/z
```

is not.

SEE ALSO

mkdir(1), mv(1).

WARNINGS

Only the super-user can use *mvdir*.

NAME

nawk – pattern scanning and processing language

SYNOPSIS

nawk [**-F** *re*] [*parameter...*] [*'prog'*] [**-f** *progfile*] [*file...*]

DESCRIPTION

nawk is a new version of *awk* that provides capabilities unavailable in previous versions. This version will become the default version of *awk* in the next major UNIX system release.

The **-F** *re* option defines the input field separator to be the regular expression *re*.

Parameters, in the form x=... y=... may be passed to *nawk,* where x and y are *nawk* built-in variables (see list below).

nawk scans each input *file* for lines that match any of a set of patterns specified in *prog*. The *prog* string must be enclosed in single quotes (') to protect it from the shell. For each pattern in *prog* there may be an associated action performed when a line of a *file* matches the pattern. The set of pattern-action statements may appear literally as *prog* or in a file specified with the **-f** *progfile* option.

Input files are read in order; if there are no files, the standard input is read. The file name - means the standard input. Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is normally made up of fields separated by white space. (This default can be changed by using the FS built-in variable or the **-F** *re* option.) The fields are denoted **$1, $2,** . . . ; **$0** refers to the entire line.

A pattern-action statement has the form:

    pattern { action }

Either pattern or action may be omitted. If there is no action with a pattern, the matching line is printed. If there is no pattern with an action, the action is performed on every input line.

Patterns are arbitrary Boolean combinations (**!**, **||**, **&&**, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

    expression relop expression
    expression matchop regular expression

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

    var **in** array,

or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line has been read and after the last input line has been read respectively.

Regular expressions are as in *egrep* [see *grep*(1)]. In patterns they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second pattern.

A regular expression may be used to separate fields by using the **-F** *re* option or by assigning the expression to the built-in variable FS. The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

Other built-in variables include:

ARGC	command line argument count
ARGV	command line argument array
FILENAME	name of the current input file
FNR	ordinal number of the current record in the current file
FS	input field separator regular expression (default blank)
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default **%.6g**)
OFS	output field separator (default blank)
ORS	output record separator (default new-line)
RS	input record separator (default new-line)

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

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
do statement while ( conditional )
for ( expression ; conditional ; expression ) statement
for ( var in array ) statement
delete array[subscript]
break
continue
{ [ statement ] ... }
expression      # commonly variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next            # skip remaining patterns on this input line
exit [expr]     # skip the rest of the input; exit status is expr
return [expr]
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators +,

-, *, /, %, and concatenation (indicated by a blank). The **C** operators ++, --, +=, -=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]), or fields. Variables are initialized to the null string or zero. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").

The **print** statement prints its arguments on the standard output, or on a file if *>expression* is present, or on a pipe if | *cmd* is present. The arguments are separated by the current output field separator and terminated by the output record separator. The **printf** statement formats its expression list according to the format [see *printf*(3S) in the *Programmer's Reference Manual*].

nawk has a variety of built-in functions: arithmetic, string, input/output, and general.

The arithmetic functions are: *atan2, cos, exp, int, log, rand, sin, sqrt,* and *srand. int* truncates its argument to an integer. *rand* returns a random number between 0 and 1. *srand* (expr) sets the seed value for *rand* to *expr* or uses the time of day if *expr* is omitted.

The string functions are:

gsub(for, repl, in)	behaves like *sub* (see below), except that it replaces successive occurrences of the regular expression (like the *ed* global substitute command).
index(s, t)	returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.
length(s)	returns the length of its argument taken as a string, or of the whole line if there is no argument.
match(s, re)	returns the position in string *s* where the regular expression *re* occurs, or 0 if it does not occur at all. RSTART is set to the starting position (which is the same as the returned value), and RLENGTH is set to the length of the matched string.
split(s, a, fs)	splits the string *s* into array elements *a*[*1*], *a*[2], *a*[*n*], and returns *n*. The separation is done with the regular expression *fs* or with the field separator FS if *fs* is not given.
sprintf(fmt, expr, expr, ...)	formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.
sub(for, repl, in)	substitutes the string *repl* in place of the first instance of the regular expression *for* in string *in* and returns the number of substitutions. If *in* is omitted, *nawk* substitutes in the current record (**$0**).
substr(s, m, n)	returns the *n*-character substring of *s* that begins at position *m*.

The input/output and general functions are:

close(filename)	closes the file or pipe named *filename*.
cmd ¦ *getline*	pipes the output of *cmd* into *getline*; each successive call to *getline* returns the next line of output from *cmd*.
getline	sets **$0** to the next input record from the current input file.
getline <file	sets **$0** to the next record from *file*.
getline var	sets variable *var* instead.
getline var <file	sets *var* from the next record of *file*.
system(cmd)	executes *cmd* and returns its exit status.

All forms of *getline* return 1 for successful input, 0 for end of file, and −1 for an error.

nawk also provides user-defined functions. Such functions may be defined (in the pattern position of a pattern-action statement) as

```
function name(args,...) { stmts }
func name(args,...) { stmts }
```

Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The **return** statement may be used to return a value.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN { FS = ",[ \t]*¦[ \t]+" }
      { print $2, $1 }
```

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

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

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

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

Simulate *echo*(1):

```
BEGIN {
        for (i = 1; i < ARGC; i++)
                printf "%s", ARGV[i]
                printf "\n"
                exit
                }
```

Print file, filling in page numbers starting at 5:

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

command line: **nawk -f program n=5 input**

SEE ALSO

grep(1), sed(1).

lex(1), printf(3S) in the *Programmer's Reference Manual.*

Programmer's Guide.

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

ncheck – generate path names from i-numbers

SYNOPSIS

/etc/ncheck [**-i** i-numbers] [**-a**] [**-s**] [file-system]

DESCRIPTION

The *ncheck* command with no arguments generates a path name vs. i-number list of all files on a set of default file systems (see */etc/checklist*). Names of directory files are followed by **/.**.

The options are as follows:

-i limits the report to only those files whose i-numbers follow.

-a allows printing of the names **.** and **..**, which are ordinarily suppressed.

-s limits the report to special files and files with set-user-ID mode. This option may be used to detect violations of security policy.

File system must be specified by the file system's special file.

The report should be sorted so that it is more useful.

SEE ALSO

fsck(1M), sort(1).

DIAGNOSTICS

If the file system structure is not consistent, **??** denotes the "parent" of a parentless file, and a path name beginning with **...** denotes a loop.

NAME

newform – change the format of a text file

SYNOPSIS

newform [**-s**] [**-i**tabspec] [**-o**tabspec] [**-b**n] [**-e**n] [**-p**n] [**-a**n] [**-f**] [**-c**char] [**-l**n] [files]

DESCRIPTION

The *newform* command 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 "**-e**15 **-l**60" will yield results different from "**-l**60 **-e**15". Options are applied to all *files* on the command line.

-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

-i*tabspec* 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**.

-o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **-i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** means that no spaces will be converted to tabs on output.

-b*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **-l***n*). 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

-e*n* Same as **-b***n* except that characters are truncated from the end of the line.

-p*n* Prefix *n* characters (see **-c***k*) 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.

-a*n* Same as **-p***n* 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**.

-c*k* Change the prefix/append character to *k*. Default character for *k* is a space.

-l*n* 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).

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.

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*newform* was called with a bad option.
not -s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

0 – normal execution
1 – for any error

SEE ALSO

csplit(1), tabs(1).
fspec(4) in the *Programmer's Reference Manual.*

BUGS

The *newform* command 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

The *newgrp* command changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (PS1) other than **$** (default) and has not exported PS1. After an invocation of *newgrp* , successful or not, their PS1 will now be set to the default prompt string **$**. Note that the shell command *export* [see *sh*(1)] is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry. This is a way to exit the effect of an earlier *newgrp* command.

If the first argument to *newgrp* is a –, the environment is changed to what would be expected if the user actually logged in again as a member of the new group.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), sh(1).
group(4), passwd(4), environ(5) in the *Programmer's Reference Manual.*

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

The *news* command 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."

-a option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

-n option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

-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) in the *Programmer's Reference Manual.*

NAME

nice – run a command at low priority

SYNOPSIS

nice [-increment] command [arguments]

DESCRIPTION

The *nice* command 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) in the *Programmer's Reference Manual.*

DIAGNOSTICS

The *nice* command 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

The *nl* command 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):

Line contents	*Start of*
\:\:\:	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:

-h*type* Same as **-b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

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).

-f*type* Same as **-b***type* except for footer. Default for logical page footer is **n** (no lines numbered).

-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**.

-p Do not restart numbering at logical page delimiters.

-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**.

-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: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes supressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

-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
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

NAME

nlsadmin – network listener service administration

SYNOPSIS

nlsadmin -x
nlsadmin [options] net_spec

DESCRIPTION

nlsadmin administers the network listener process(es) on a machine. Each network has a separate instance of the network listener process associated with it; each instance (and thus, each network) is configured separately. The listener process "listens" to the network for service requests, accepts requests when they arrive, and spawns servers in response to those service requests. The network listener process will work with any network (more precisely, with any transport provider) that conforms to the transport provider specification.

The listener supports two classes of service: a general listener service, serving processes on remote machines, and a terminal login service, for terminals connected directly to a network. The terminal login service provides networked access to this machine in a form suitable for terminals connected directly to the network. However, this direct terminal service requires special associated software, and is only available with some networks (for example, the AT&T STARLAN network).

nlsadmin can establish a listener process for a given network, configure the specific attributes of that listener, and start and kill the listener process for that network. *nlsadmin* can also report on the listener processes on a machine, either individually (per network) or collectively.

The following list shows how to use *nlsadmin*. In this list, *net_spec* represents a particular listener process. Specifically, *net_spec* is the relative path name of the entry under **/dev** for a given network (that is, a transport provider). Changing the list of services provided by the listener produces immediate changes, while changing an address on which the listener listens has no effect until the listener is restarted. The following combination of *options* can be used.

no options	will give a brief usage message.
-x	will report the status of all of the listener processes installed on this machine.
net_spec	will print the status of the listener process for *net_spec*.
-q *net_spec*	will query the status of the listener process for the specified network, and will reflect the result of that query in its exit code. If a listener process is active, *nlsadmin* will exit with a status of 0; if no process is active, the exit code will be 1; the exit code will be greater than 1 in case of error.
-v *net_spec*	will print a verbose report on the servers associated with *net_spec*, giving the service code, status, command, and comment for each. It also specifies the

uid the server will run as, and the list of modules to be pushed, if any, before the server is started.

-z *service_code net_spec*
: will print a report on the server associated with *net_spec* that has service code *service_code*, giving the same information as in the **-v** option.

-q -z *service_code net_spec*
: will query the status of the service with service code *service_code* on network *net_spec*, and will exit with a status of 0 if that service is enabled, 1 if that service is disabled, and greater than 1 in case of error.

-l *addr net_spec*
: will change or set the address on which the listener listens (the general listener service). This is the address generally used by remote processes to access the servers available through this listener (see the **-a** option, below). *addr* is the transport address on which to listen and is interpreted using a syntax that allows for a variety of address formats. By default *addr* is interpreted as the symbolic ASCII representation of the transport address. An *addr* preceded by a **x** will let you enter an address in hexadecimal notation. Note that *addr* must appear as a single word to the shell and must be quoted if it contains any blanks.

If *addr* is just a dash ("-"), *nlsadmin* will report the address currently configured, instead of changing it.

A change of address will not take effect until the next time the listener for that network is started.

-t *addr net_spec*
: will change or set the address on which the listener listens for requests for terminal service, but is otherwise similar to the **-l** option above. A terminal service address should not be defined unless the appropriate remote login software is available; if such software is available, it must be configured as service code 1 (see the **-a** option, below).

-i *net_spec*
: will initialize or change a listener process for the network specified by *net_spec*; that is, it will create and initialize the files required by the listener. Note that the listener should only be initialized once for a given network, and that doing so does not actually invoke the listener for that network. The listener must be initialized before assigning addressing or services.

[**-m**] **-a** *service_code* [**-p** *modules*] [**-w** *id*] **-c** *cmd* **-y** *comment net_spec*
: will add a new service to the list of services available through the indicated listener. *service_code* is the

code for the service, *cmd* is the command to be invoked in response to that service code, comprised of the full path name of the server and its arguments, and *comment* is a brief (free-form) description of the service for use in various reports. Note that *cmd* must appear as a single word to the shell, so if arguments are required the *cmd* and its arguments must be surrounded by quotes. Similarly, the *comment* must also appear as a single word to the shell. When a service is added, it is initially enabled (see the **-e** and **-d** options, below).

If the **-m** option is specified, the entry will be marked as an administrative entry. Service codes 1 through 100 are reserved for administrative entries, which are those that require special handling internally. In particular, code 1 is assigned to the remote login service, which is the service automatically invoked for connections to the terminal login address.

The **-m** option used with the **-a** option indicates that special handling internally is required for those servers added with the **-m** set. This internal handling is in the form of code embedded on the listener process.

If the **-p** option is specified, then *modules* will be interpreted as a list of STREAMS modules for the listener to push before starting the service being added. The modules are pushed in the order they are specified. *modules* should be a comma-separated list of modules, with no white space included.

If the **-w** option is specified, then *id* is interpreted as the user name from **/etc/passwd** that the listener should look up. From the user name, the listener should obtain the user ID, the group ID, and the home directory for use by the server. If **-w** is not specified, the default is to use the user ID **listen.**

A service must explicitly be added to the listener for each network on which that service is to be available. This operation will normally be performed only when the service is installed on a machine, or when populating the list of services for a new network.

-r *service_code net_spec*
will remove the entry for the *service_code* from that listener's list of services. This will normally be performed only in conjunction with the de-installation of a service from a machine.

-e *service_code net_spec*
-d *service_code net_spec*
will enable or disable (respectively) the service indicated by *service_code* for the specified network. The service must have previously been added to the listener for that network (see the **-a** option, above). Disabling a service will cause subsequent service requests for that service to be denied, but the processes from any prior service requests that are still running will continue unaffected.

-s *net_spec*
-k *net_spec*
will start and kill (respectively) the listener process for the indicated network. These operations will normally be performed as part of the system startup and shutdown procedures. Before a listener can be started for a particular network, it must first have been initialized, and an address must be defined for the general listener service (see the **-i** and **-l** options, above). When a listener is killed, processes that are still running as a result of prior service requests will continue unaffected.

The listener runs as user ID **root**, with group ID **sys**. A special ID, user ID **listen** and group ID **adm**, should be entered in the **/etc/passwd** file as a default ID for servers. The listener always uses as its home directory **/usr/net/nls**, which is concatenated with *net_spec* to determine the location of the listener configuration information for each network. The home directory specified in the **/etc/passwd** entry for **listener** will used by servers that run as ID **listen**.

nlsadmin may be invoked by any user to generate reports, but all operations that affect a listener's status or configuration are restricted to the super-user.

FILES

/usr/net/nls/*net_spec*

SEE ALSO

Network Programmer's Guide

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup command [arguments]

DESCRIPTION

The *nohup* command executes *command* with hangups and quits ignored. If output is not re-directed by the user, both standard output and standard error are sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out.**

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file.* If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored [see *sh*(1)]:

```
nohup file &
```

An example of what the contents of *file* could be is:

```
sort ofile > nfile
```

SEE ALSO

chmod(1), nice(1), sh(1),
signal(2) in the *Programmer's Reference Manual.*

WARNINGS

In the case of the following command

```
nohup command1; command2
```

nohup applies only to command1. The command

```
nohup (command1; command2)
```

is syntactically incorrect.

NAME

nsquery – Remote File Sharing name server query

SYNOPSIS

nsquery [**-h**] [*name*]

DESCRIPTION

The *nsquery* command provides information about resources available to the host from both the local domain and from other domains; all resources are reported, regardless of whether the host is authorized to access them. When used with no options, *nsquery* identifies all resources in the domain that have been advertised as sharable. A report on selected resources can be obtained by specifying *name*, where *name* is:

nodename	The report will include only those resources available from *nodename*.
domain.	The report will include only those resources available from *domain*.
domain.nodename	The report will include only those resources available from *domain.nodename*.

When the name does not include the delimiter ".", it will be interpreted as a *nodename* within the local domain. If the name ends with a delimiter ".", it will be interpreted as a domain name.

The information contained in the report on each resource includes its advertised name (domain.resource), the read/write permissions, the server (nodename.domain) that advertised the resource, and a brief textual description.

When *-h* is used, the header is not printed.

A remote domain must be listed in your **rfmaster** file in order to query that domain.

EXIT STATUS

If no entries are found when *nsquery* is executed, the report header is printed.

ERRORS

If your host cannot contact the domain name server, an error message will be sent to standard error.

SEE ALSO

adv(1M), unadv(1M).
rfmaster(4) in the *Programmer's Reference Manual.*

NAME

od – octal dump

SYNOPSIS

od [**-bcdosx**] [file] [[+]offset[**.**][**b**]]

DESCRIPTION

The *od* command 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.

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [-] [**-f**] name ...

pcat name ...

unpack name ...

DESCRIPTION

The *pack* command 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*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

The *pack* command 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.

The *pack* command 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.

The *pcat* command does for packed files what *cat*(1) does for ordinary files, except that *pcat* cannot be used as a filter. 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
```

The *pcat* command 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.

SEE ALSO

cat(1).

NAME

passmgmt – password files management

SYNOPSIS

passmgmt -a [**-c** *comment*] [**-h** *homedir*] [**-u** *uid* [**-o**]] [**-g** *gid*] [**-s** *shell*] *name*

passmgmt -m [**-c** *comment*] [**-h** *homedir*] [**-u** *uid* [**-o**]] [**-g** *gid*] [**-s** *shell*] [**-l** *logname*] *name*

passmgmt -d *name*

DESCRIPTION

passmgmt -a adds an entry for user *name* to the login password files. This command does not create any directory for the new user and the new login remains locked until the *passwd* command is executed to set the password.

passmgmt -m modifies the entry for user *name* in the login password files. All the fields (except the password field) in the password entry and the name field in the shadow password entry can be modified by this command. Only fields entered on the command line will be modified.

passmgmt -d deletes the entry for user *name* from the login password files. It will not remove any files that the user owns on the system; they must be removed manually.

The following options are available:

-c *comment*	A short description of the login. It is limited to a maximum of 128 characters and defaults to an empty field.
-h *homedir*	Default home directory of the user. It is limited to a maximum of 256 characters and defaults to **/usr/***name*.
-u *uid*	UID of the *name*. This number must range from 0 to the maximum non-negative value for the system. It defaults to the next available UID greater than 100. Without the **-o** option, it enforces the uniqueness of a UID.
-o	This option allows a UID to be non-unique.
-g *gid*	GID of the *name*. This number must range from 0 to the maximum non-negative value for the system. The default is 1.
-s *shell*	Login shell for *name*. It should be the full pathname of the program that will be executed when the user logs in. The maximum size of *shell* is 256 characters. It defaults to **/bin/sh**.
-l *logname*	Change *logname*. This option changes the *name* to *logname*.
name	The login *name* of the user. It must be unique and alphanumeric.

The total size of each login entry for the password and shadow files, whether existing or new, is limited to a maximum of 511 bytes.

FILES

/etc/passwd, /etc/shadow, /etc/opasswd, /etc/oshadow

SEE ALSO

passwd(1), passwd(1M).

passwd(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

The **passmgmt** command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid command syntax. Usage message of the **passmgmt** command will be displayed.
3	Invalid argument provided to option.
4	UID in use.
5	Inconsistent password files (e.g., *name* is in the **/etc/passwd** file and not in the **/etc/shadow** file, or vice versa).
6	Unexpected failure. Password files unchanged.
7	Unexpected failure. Password file(s) missing.
8	Password file(s) busy. Try again later.
9	*name* does not exist (if **-m** or **-d** is specified), already exists (if **-a** is specified), or *logname* already exists (if **-m -l** is specified).

WARNING

Do not use a colon in the comment (**-c** option) because it will be interpreted as a field separator.

NAME

passwd – change login password

SYNOPSIS

passwd [name]

passwd -s [name]

DESCRIPTION

The *passwd* command changes the password associated with the login *name*. The **-s** option shows password attributes for the login *name*.

The format of the display will be:

name status mm/dd/yy min max

or, if password aging information is not present,

name status

where:

name	The login ID of the user.
status	The password status of *name*: "PS" stands for passworded or locked, "LK" stands for locked, and "NP" stands for no password.
mm/dd/yy	The date password was last changed for *name*.
min	The minimum number of days required between password changes for *name*.
max	The maximum number of days the password is valid for *name*.

Ordinary users may change only their own password or display the password attributes that correspond to their login *name*.

The *passwd* command prompts ordinary users for their old password, if any. It then prompts for the new password twice. When the old password is entered, *passwd* checks to see if it has "aged" sufficiently. If "aging" is insufficient, *passwd* terminates; see *passwd*(4).

Assuming aging is sufficient, a check is made to ensure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical, the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

Each password must have at least six characters. Only the first eight characters are significant.

Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" refers to all uppercase or lowercase letters.

Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an uppercase letter and its corresponding lower case letter are equivalent.

New passwords must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

Super-users [e.g., real and effective *uid* equal to zero, [see *id*(1M) and *su*(1M)] may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

login(1).

crypt(3C), passwd(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

The **passwd** command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid combination of options.
3	Unexpected failure. Password file unchanged.
4	Unexpected failure. Password file(s) missing.
5	Password file(s) busy. Try again later.

NAME

passwd – change login password and password attributes

SYNOPSIS

passwd [name]

passwd -l [**-f**] [**-x** *max*] [**-n** *min*] name

passwd -d [**-f**] [**-x** *max*] [**-n** *min*] name

passwd -s [**-a**]

passwd -s [name]

DESCRIPTION

This command changes or installs login passwords and password attributes associated with the login *name*. The options to *passwd* are:

-l Locks password entry for *name*.

-d Deletes password for *name*. The login *name* will not be prompted for password.

-x Set maximum field for *name*. The *max* field contains the number of days that the password is valid for *name*. The aging for *name* will be turned off if *max* is set to 0.

-n Set minimum field for *name*. The *min* field contains the minimum number of days between password changes for *name*.

-s Shows password attributes for *name*. The format of the display will be:

name status mm/dd/yy min max

or, if password aging information is not present,

name status

name is the login ID of the user. *status* is the password status of *name*. "PS" stands for passworded or locked, "LK" stands for locked, and "NP" stands for no password. *mm/dd/yy* is the date password was last changed of *name*. *min* is the minimum number of days required between password changes of *name*. *max* is the maximum number of days the password is valid of *name*.

-a Show password attributes for all entries. Use only with **-s** option; *name* must not be provided.

-f Force the user to change password at the next login by expiring the password for *name*.

Privileged users may change any password; hence, *passwd* does not prompt privileged users for the old password. Privileged users are not forced to comply with password aging and password construction requirements. A privileged user can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

id(1M), login(1), passmgmt(1M), passwd(1), su(1M).
crypt(3C), passwd(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

The *passwd* command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid combination of options.
3	Unexpected failure. Password file unchanged.
4	Unexpected failure. Password file missing.
5	Password file(s) busy. Try again later.
6	Invalid argument to option.

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste -dlist file1 file2 ...
paste -s [-dlist**]** 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* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. 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

cut(1), grep(1), pr(1).

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

pg – file perusal filter for CRTs

SYNOPSIS

pg [-*number*] [-p *string*] [-cefns] [+*linenumber*] [+/*pattern*/] [files...]

DESCRIPTION

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a CRT. (The file name – and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo*(4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

–number
: An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

-p *string*
: Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

-c Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo*(4) data base.

-e Causes *pg* *not* to pause at the end of each file.

-f Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The *-f* option inhibits *pg* from splitting lines.

-n Normally, commands must be terminated by a <*newline*> character. This option causes an automatic end of command as soon as a command letter is entered.

-s Causes *pg* to print all messages and prompts in standout mode (usually inverse video).

+linenumber
: Start up at *linenumber*.

+/pattern/
: Start up at the first line containing the regular expression pattern.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<*newline*> or <*blank*>
: This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
: Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L** Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(1) are available. They must always be terminated by a <*newline*>, even if the *-n* option is specified.

i/pattern/
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*
i?pattern?
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w** Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a *<newline>*, even if the *-n* option is specified.

h Help by displaying an abbreviated summary of available commands.

q or **Q** Quit *pg*.

!*command*
Command is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a *<newline>*, even if the *-n* option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK, DEL,** and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

/usr/lib/terminfo/?/*	terminal information database
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

ed(1), grep(1).
terminfo(4) in the *Programmer's Reference Manual* .

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options, terminal settings may not be restored correctly.

NAME

pr – print files

SYNOPSIS

pr [[**-column**] [**-w**width] [**-a**]] [**-e**ck] [**-i**ck] [**-drtfp**] [+page] [**-n**ck] [**-o**offset] [**-l**length] [**-s**separator] [**-h**header] [file ...]

pr [[**-m**] [**-w**width]] [**-e**ck] [**-i**ck] [**-drtfp**] [+page] [**-n**ck] [**-o**offset] [**-l**length] [**-s**separator] [**-h**header] file1 file2 ...

DESCRIPTION

pr is used to format and print the contents of a file. If *file* is **-**, or if no files are specified, *pr* assumes standard input. *pr* prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, the date and time that the file was last modified, and the name of the file. Page length is 66 lines, which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with **-h**), and 2 blank lines; the trailer is 5 blank lines. For single column output, line width may not be set and is unlimited. For multicolumn output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

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 *separator* character.

Either *-column* or **-m** should be used to produce multi-column output. **-a** should only be used with *-column* and not **-m**.

Command line options are

+page	Begin printing with page numbered *page* (default is 1).
-column	Print *column* columns of output (default is 1). Output appears as if **-e** and **-i** are turned on for multi-column output. May not use with **-m**.
-a	Print multi-column output across the page one line per column. *columns* must be greater than one. If a line is too long to fit in a column, it is truncated.
-m	Merge and print all files simultaneously, one per column. The maximum number of files that may be specified is eight. If a line is too long to fit in a column, it is truncated. May not use with *-column*.
-d	Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.
-e*ck*	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).

-i*ck* 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).

-n*ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of single column 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).

-w*width* Set the width of a line to *width* character positions (default is 72). This is effective only for multi-column output (*-column* and **-m**). There is no line limit for single column output.

-o*offset* Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

-l*length* Set the length of a page to *length* lines (default is 66). **-l0** is reset to **-l66**. When the value of *length* is 10 or less, **-t** appears to be in effect since headers and trailers are suppressed. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When **-l***length* is used and *length* exceeds 10, then *length*-10 lines are left per page for user-supplied text. When *length* is 10 or less, header and trailer output is omitted to make room for user-supplied text.

-h *header* Use *header* as the text line of the header to be printed instead of the file name. **-h** is ignored when **-t** is specified or **-l***length* is specified and the value of *length* is 10 or less. (**-h is the only** *pr* option requiring space between the option and argument.)

-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 single 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 files that will not open.

-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. Use of **-t** overrides the **-h** option.

-s*separator*

Separate columns by the single character *separator* instead of by the appropriate number of spaces (default for *separator* is a tab). Prevents truncation of lines on multicolumn output unless **-w** is specified.

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Copy **file1** to **file2**, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr -e9 -t <file1 >file2
```

Print **file1** and **file2** simultaneously in a two-column listing with no header or trailer where both columns have line numbers:

```
pr -t -n file1 | pr -t -m -n file2 -
```

FILES

/dev/tty* If standard output is directed to one of the special files **/dev/tty***, then other output directed to this terminal is delayed until standard output is completed. This prevents error messages from being interspersed throughout the output.

SEE ALSO

cat(1), pg(1).

NAME

profiler: prfld, prfstat, prfdc, prfsnap, prfpr – UNIX system profiler

SYNOPSIS

/etc/prfld [system_namelist]
/etc/prfstat on
/etc/prfstat off
/etc/prfdc file [period [off_hour]]
/etc/prfsnap file
/etc/prfpr file [cutoff [system_namelist]]

DESCRIPTION

The *prfld, prfstat, prfdc, prfsnap,* and *prfpr* routines form a system of programs to facilitate an activity study of the UNIX operating system.

The *prfld* program is used to initialize the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from *system_namelist*.

The *prfstat* program is used to enable or disable the sampling mechanism. Profiler overhead is less than 1% as calculated for 500 text addresses. *Prfstat* will also reveal the number of text addresses being measured.

The *prfdc* and *prfsnap* programs perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. *Prfdc* will store the counters into *file* every *period* minutes and will turn off at *off_hour* (valid values for *off_hour* are **0–24**). *Prfsnap* collects data at the time of invocation only, appending the counter values to *file*.

The *prfpr* program formats the data collected by *prfdc* or *prfsnap*. Each text address is converted to the nearest text symbol (as found in *system_namelist*) and is printed if the percent activity for that range is greater than *cutoff*.

FILES

/dev/prf	interface to profile data and text addresses
/unix	default for system namelist file

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 controlling terminal. Output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options.*

Options accept names or lists as arguments. Arguments can be either separated from one another by commas or enclosed in double quotes and separated from one another by commas or spaces. Values for *proclist* and *grplist* must be numeric.

The *options* are given in descending order according to volume and range of information provided:

-e	Print information about **e**very process now running.
-d	Print information about all processes except process group leaders.
-a	Print information about **a**ll processes most frequently requested: all those except process group leaders and processes not associated with a terminal.
-f	Generate a **f**ull listing. (See below for significance of columns in a full listing.)
-l	Generate a **l**ong listing. (See the following text.)
-n *name*	Valid only for users with a real user id of *root* or a real group id of *sys*. Takes argument signifying an alternate system *name* in place of **/unix**.
-t *termlist*	List only process data associated with the terminal given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's file name (e.g., **tty04**) or, if the device's file name starts with **tty**, just the digit identifier (e.g., **04**).
-p *proclist*	List only process data whose process ID numbers are given in *proclist*.
-u *uidlist*	List only process data whose user ID number or login name is given in *uidlist*. In the listing, the numerical user ID will be printed unless you give the **-f** option, which prints the login name.
-g *grplist*	List only process data whose process group leader's ID number(s) appears in *grplist*. (A group leader is a process whose process ID number is identical to its process group ID number. A login shell is a common example of a process group leader.)

Under the **-f** option, *ps* tries to determine the command name and arguments given when the process was created by examining the user block. Failing this, the command name is printed, as it would have appeared without the **-f** option, in square brackets.

The column headings and the meaning of the columns in a *ps* listing are given in the following text; the letters **f** and **l** indicate the option (full or long, respectively) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do not determine which processes will be listed.

F	(l)	Flags (hexadecimal and additive) associated with the process

00	Process has terminated: process table entry now available.
01	A system process: always in primary memory.
02	Parent is tracing process.
04	Tracing parent's signal has stopped process: parent is waiting [*ptrace*(2)].
08	Process is currently in primary memory.
10	Process currently in primary memory: locked until an event completes.

S	(l)	The state of the process:

O	Process is running on a processor.
S	Sleeping: process is waiting for an event to complete.
R	Runnable: process is on run queue.
I	Idle: process is being created.
Z	Zombie state: process terminated and parent not waiting.
T	Traced: process stopped by a signal because parent is tracing it.
X	SXBRK state: process is waiting for more primary memory.

UID	(f,l)	The user ID number of the process owner (the login name is printed under the **-f** option).
PID	(all)	The process ID of the process (this datum is necessary in order to kill a process).
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling.
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.
SZ	(l)	The size (in pages or clicks) of the swappable process's image in main memory.

WCHAN	(l)	The address of an event for which the process is sleeping, or in SXBRK state, (if blank, the process is running).
STIME	(f)	The starting time of the process, given in hours, minutes, and seconds. (A process begun more than twenty-four hours before the *ps* inquiry is executed is given in months and days.)
TTY	(all)	The controlling terminal for the process (the message, **?**, is printed when there is no controlling terminal).
TIME	(all)	The cumulative execution time for the process.
COMMAND	(all)	The command name (the full command name and its arguments are printed under the **-f** option).

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

FILES

/dev	
/dev/sxt/*	
/dev/tty*	
/dev/xt/*	terminal ("tty") names searcher files
/dev/kmem	kernel virtual memory
/dev/swap	the default swap device
/dev/mem	memory
/etc/passwd	UID information supplier
/etc/ps_data	internal data structure
/unix	system name list

SEE ALSO

getty(1M), kill(1), nice(1).

WARNING

Things can change while *ps* is running; the snap-shot it gives is only true for a split-second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no *termlist, proclist, uidlist,* or *grplist* is specified, *ps* checks *stdin, stdout,* and *stderr* in that order, looking for the controlling terminal and will attempt to report on processes associated with the controlling terminal. In this situation, if *stdin, stdout,* and *stderr* are all redirected, *ps* will not find a controlling terminal, so there will be no report.

On a heavily loaded system, *ps* may report an *lseek*(2) error and exit. *ps* may seek to an invalid user area address: having obtained the address of a process' user area, *ps* may not be able to seek to that address before the process exits and the address becomes invalid.

ps -ef may not report the actual start of a tty login session, but rather an earlier time, when a getty was last respawned on the tty line.

NAME

pwck, grpck – password/group file checkers

SYNOPSIS

/etc/pwck [file]
/etc/grpck [file]

DESCRIPTION

The *pwck* command scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and the program-to-use-as-Shell exist. The default password file is **/etc/passwd**.

The *grpck* command verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

NAME

pwconv – install and update **/etc/shadow** with information from **/etc/passwd**

SYNOPSIS

pwconv

DESCRIPTION

The *pwconv* command creates and updates **/etc/shadow** with information from **/etc/passwd.** If the **/etc/shadow** file does not exist, this command will create **/etc/shadow** with information from **/etc/passwd.** The command populates **/etc/shadow** with the user's login name, password, and password aging information. If password aging information does not exist in **/etc/passwd** for a given user, none will be added to **/etc/shadow.** However, the "last changed" information will always be updated. The *passwd* command should be used to add or change password aging information.

If the **/etc/shadow** file does exist, the following tasks will be performed:

> Entries that are in the **/etc/passwd** file and not in the **/etc/shadow** file will be added to the **/etc/shadow** file.
>
> Entries that are in the **/etc/shadow** file and not in the **/etc/passwd** file will be removed from **/etc/passwd.**
>
> Password attributes (e.g., password and aging information) that exist in an **/etc/passwd** entry will be moved to the corresponding entry in **/etc/shadow.**

The following is the format of an entry in **/etc/passwd**:

name:passwd,aging:uid:gid:comment:homedir:shell

The following shows how changes made to **/etc/passwd** affect **/etc/shadow** and **/etc/passwd** when *pconv* is run:

name	delete old entry from **/etc/shadow,** add new entry to **/etc/shadow**
passwd	update entry in **/etc/shadow,** place an **x** in **/etc/passwd**
aging	update entry in **/etc/shadow,** clear aging field in **/etc/passwd**
uid	ignore—no change to **/etc/shadow**
gid	ignore—no change to **/etc/shadow**
comment	ignore—no change to **/etc/shadow**
homedir	ignore—no change to **/etc/shadow**
shell	ignore—no change to **/etc/shadow**

pwconv is a privileged system command that cannot be executed by ordinary users.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

passmgmt(1M), passwd(1M).

DIAGNOSTICS

The **pwconv** command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid command syntax.
3	Unexpected failure. Conversion not done.
4	Unexpected failure. Password file(s) missing.
5	Password file(s) busy. Try again later.

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

The *pwd* command prints the path name of the working (current) directory.

SEE ALSO

cd(1).

NAME

rc0 – run commands performed to stop the operating system

SYNOPSIS

/etc/rc0

DESCRIPTION

This file is executed at each system state change that needs to have the system in an inactive state. It is responsible for those actions that bring the system to a quiescent state, traditionally called "shutdown".

One system state requires this procedure: state 0 (the system halt state). Whenever a change to this state occurs, the */etc/rc0* procedure is run. The entry in **/etc/inittab** might read:

```
s0:0:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
```

Some of the actions performed by */etc/rc0* are carried out by files in the directory **/etc/shutdown.d**. and files beginning with **K** in **/etc/rc0.d.** These files are executed in ASCII order (see FILES below for more information), terminating some system service. The combination of commands in */etc/rc0* and files in **/etc/shutdown.d** and **/etc/rc0.d** determines how the system is shut down.

The recommended sequence for */etc/rc0* is:

Stop System Services and Daemons.

> Various system services (such as Remote File Sharing or LP Spooler) are gracefully terminated.
>
> When new services are added that should be terminated when the system is shut down, the appropriate files are installed in **/etc/shutdown.d** and **/etc/rc0.d.**

Terminate Processes

> SIGTERM signals are sent to all running processes by *killall*(1M). Processes stop themselves cleanly if sent SIGTERM.

Kill Processes

> SIGKILL signals are sent to all remaining processes; no process can resist SIGKILL.
>
> At this point the only processes left are those associated with */etc/rc0 and processes 0 and 1,* which are special to the operating system.

Unmount All File Systems

> Only the root file system (/) remains mounted.

FILES

The execution by **/bin/sh** of any files in **/etc/shutdown.d** occurs in ASCII sort-sequence order. See *rc2*(1M) for more information.

SEE ALSO

killall(1M), rc2(1M), shutdown(1M).

NAME

rc2 – run commands performed for multiuser environment

SYNOPSIS

/etc/rc2

DESCRIPTION

This file is executed via an entry in **/etc/inittab** and is responsible for those initializations that bring the system to a ready-to-use state, traditionally state 2, called the "multiuser" state.

The actions performed by **/etc/rc2** are found in files in the directory **/etc/rc.d** and files beginning with **S** in **/etc/rc2.d**. These files are executed by **/bin/sh** in ASCII sort–sequence order (see FILES for more information). When functions are added that need to be initialized when the system goes multiuser, an appropriate file should be added in **/etc/rc2.d**.

The functions done by **/etc/rc2** command and associated **/etc/rc2.d** files include:

Setting and exporting the TIMEZONE variable.

Setting up and mounting the user (**/usr**) file system.

Cleaning up (remaking) the **/tmp** and **/usr/tmp** directories.

Loading the network interface and ports cards with program data and starting the associated processes.

Starting the *cron* daemon by executing **/etc/cron**.

Cleaning up (deleting) uucp locks status and temporary files in the **/usr/spool/uucp** directory.

Other functions can be added, as required, to support the addition of hardware and software features.

EXAMPLES

The following are prototypical files found in **/etc/rc2.d**. These files are prefixed by an **S** and a number indicating the execution order of the files.

MOUNTFILESYS

```
#   Set up and mount file systems

cd /
/etc/mountall /etc/fstab
```

RMTMPFILES

```
# clean up /tmp
rm -rf /tmp
mkdir /tmp
chmod 777 /tmp
chgrp sys /tmp
chown sys /tmp
```

uucp

```
#   clean-up uucp locks, status, and temporary files

rm -rf /usr/spool/locks/*
```

The file **/etc/TIMEZONE** is included early in */etc/rc2*, thus establishing the default time zone for all commands that follow.

FILES

Here are some hints about files in **/etc/rc.d**:

The order in which files are executed is important. Since they are executed in ASCII sort-sequence order, using the first character of the file name as a sequence indicator will help keep the proper order. Thus, files starting with the following characters would be:

[0-9].	very early
[A-Z].	early
[a-n].	later
[o-z].	last

3.mountfs

Files in **/etc/rc.d** that begin with a dot (.) will not be executed. This feature can be used to hide files that are not to be executed for the time being without removing them. The command can be used only by the super-user.

Files in **/etc/rc2.d** must begin with an **S** or a **K** followed by a number and the rest of the file name. Upon entering run level 2, files beginning with **S** are executed with the **start** option; files beginning with **K**, are executed with the **stop** option. Files beginning with other characters are ignored.

SEE ALSO

shutdown(1M).

NAME

relogin – rename login entry to show current layer

SYNOPSIS

/usr/lib/layersys/relogin [-s] [line]

DESCRIPTION

The *relogin* command changes the terminal *line* field of a user's *utmp*(4) entry to the name of the windowing terminal layer attached to standard input. *write*(1) messages sent to this user are directed to this layer. In addition, the *who*(1) command will show the user associated with this layer. The *relogin* command may only be invoked under *layers*(1).

relogin is invoked automatically by *layers*(1) to set the *utmp*(4) entry to the terminal line of the first layer created upon startup and to reset the *utmp*(4) entry to the real line on termination. It may be invoked by a user to designate a different layer to receive *write*(1) messages.

-s	Suppress error messages.
line	Specifies which *utmp*(4) entry to change. The *utmp*(4) file is searched for an entry with the specified *line* field. That field is changed to the line associated with the standard input. (To learn what lines are associated with a given user, say **jdoe**, type **ps -f -u jdoe** and note the values shown in the **TTY** field [see *ps*(1)].

FILES

/etc/utmp	data base of users versus terminals

EXIT STATUS

Returns **0** upon successful completion, **1** otherwise.

SEE ALSO

layers(1), mesg(1), ps(1), who(1), write(1).
utmp(4) in the *Programmer's Reference Manual.*

NOTES

If *line* does not belong to the user issuing the *relogin* command or standard input is not associated with a terminal, *relogin* will fail.

NAME

removepkg – remove installed package

SYNOPSIS

removepkg [software_package]

DESCRIPTION

The *removepkg* command will remove the software package specified as an arguement to *removepkg* or will remove the software package the user selects if no argument is given to *removepkg*.

If an argument is specified, *removepkg* will search the list of previously installed packages and remove the first name it matchs. If no name is matched, the user is given an error message.

If no argument is specified, *removepkg* will query the user, via a menu, which package to remove.

You will need to be *root* to remove some packages.

LIMITATIONS

You must envoke *removepkg* on the console.

SEE ALSO

displaypkg(1), installpkg(1).

NAME

restore – restore file to original directory

SYNOPSIS

restore [**-c**] [**-i**] [**-o**] [**-t**] [**-d** <device>] [pattern [pattern]...]

DESCRIPTION

-c complete restore. All files on the tape are restored.

-i gets the index file off of the medium. This only works when the archive was created using backup. The output is a list of all the files on the medium. No files are actually restored.

-o overwrite existing files. If the file being restored already exists it will not be restored unless this option is specified.

-t indicates that the tape device is to be used. MUST be used with the **-d** option when restoring from tape.

-d *<device>* is the raw device to be used. It defaults to the 1.2M floppy (**/dev/rdsk/f0q15d**).

One or more patterns can be specified. These patterns are matched against the files on the tape. When a match is found, the file is restored. Since backups are done using full pathnames, the file is restored to its original directory. Metacharacters can be used to match multiple files. Patterns should be in quotes to prevent the characters from being expanded before they are passed to the command. If no patterns are specified, it defaults to restoring all files. If a pattern does not match any file on the tape, a message is printed.

When end of medium is reached, the user is prompted for the next media. The user can exit at this point by typing "q". (This may cause files to be corrupted if a file happens to span a medium.) In general, quitting in the middle is not a good idea.

If the file already exists and an attempt is made to restore it without the **-o** option, the file name will be printed on the screen followed by a question mark.

In order for multi-volume restores to work correctly, the raw device MUST be used.

SEE ALSO

qt(7).

NAME

rfadmin – Remote File Sharing administration

SYNOPSIS

rfadmin

rfadmin -[**ar**] *domain.nodename*

rfadmin -[**pq**]

rfadmin -o *option*

DESCRIPTION

rfadmin is primarily used to add and remove computers and their associated authentication information from a *domain*/**passwd** file on a Remote File Sharing primary domain name server. It is also used to transfer domain name server responsibilities from one machine to another. Used with no options, **rfadmin** returns the *domain.nodename* of the current domain name server for the local domain. Other options let you check if RFS is running and turn on the RFS loop back feature.

rfadmin can only be used to modify domain files on the primary domain name server (**-a** and **-r** options). If domain name server responsibilities are temporarily passed to a secondary domain name server, that computer can use the **-p** option to pass domain name server responsibility back to the primary. **rfadmin** can be used on any computer with no options or with the **q** or **o** options. To print information about the current domain name server, the user must have **root** permissions to use the command.

-a *domain.nodename* Used to add a computer to the member list of the domain that is served by this primary domain name server. The computer's name must be of the form *domain.nodename*. This command creates an entry for *nodename* in the *domain*/**passwd** file, which has the same format as **/etc/passwd**, and prompts for an initial authentication password. The password prompting process conforms with that of **passwd**(1).

-r *domain.nodename* Used to remove a computer from its domain by removing it from the *domain*/passwd file.

-p Used to pass the domain name server responsibilities back to a primary or to a secondary name server.

-q Prints a message that will tell you whether or not RFS is running.

-o *option* Lets you set RFS system options, by replacing *option* with one of the following:

loopback - Enables loop back facility for your computer. When this is set, you can mount a resource that is advertised from your own computer. This is used for testing applications in RFS when only one computer is available. Loop back is off by default.

noloopback - Turns off the loop back facility for your computer. This is the default.

ERRORS

When used with the **-a** option, if *domain.nodename* is not unique in the domain, an error message will be sent to standard error.

When used with the **-r** option, if (1) *domain.nodename* does not exist in the domain, (2) *domain.nodename* is defined as a domain name server, or (3) there are resources advertised by *domain.nodename*, an error message will be sent to standard error.

When used with the **-p** option to change the domain name server, if there are no backup name servers defined for *domain*, a warning message will be sent to standard error.

FILES

/usr/nserve/auth.info/*domain*/passwd
(For each *domain*, this file: is created on the primary,
should be copied to all secondaries, and should be copied to
all computers that want to do password verification of computers
in the *domain*.)

SEE ALSO

passwd(1), rfstart(1M), rfstop(1M), umount(1M).

NAME

rfpasswd – change Remote File Sharing host password

SYNOPSIS

rfpasswd

DESCRIPTION

The *rfpasswd* command updates the Remote File Sharing authentication password for a host; processing of the new password follows the same criteria as *passwd*(1). The updated password is registered at the domain name server (/usr/nserve/auth.info/*domain*/passwd) and replaces the password stored at the local host (**/usr/nserve/loc.passwd** file).

This command is restricted to the super-user.

NOTE: If you change your host password, make sure that hosts that validate your password are notified of this change. To receive the new password, hosts must obtain a copy of the *domain*/**passwd** file from the domain's primary name server. If this is not done, attempts to mount remote resources may fail!

ERRORS

If (1) the old password entered from this command does not match the existing password for this machine, (2) the two new passwords entered from this command do not match, (3) the new password does not satisfy the security criteria in *passwd*(1), (4) the domain name server does not know about this machine, or (5) the command is not run with super-user privileges, an error message will be sent to standard error. Also, Remote File Sharing must be running on your host and your domain's primary name server. A new password cannot be logged if a secondary is acting as the domain name server.

FILES

/usr/nserve/auth.info/*domain*/passwd
/usr/nserve/loc.passwd

SEE ALSO

passwd(1), rfstart(1M), rfadmin(1M).

NAME

rfstart – start Remote File Sharing

SYNOPSIS

rfstart [**-v**] [-p*primary_addr*]

DESCRIPTION

The **rfstart** command starts Remote File Sharing and defines an authentication level for incoming requests. [This command can only be used after the domain name server is set up and your computer's domain name and network specification has been defined using **dname**(1M).]

-v Specifies that verification of all clients is required in response to initial incoming mount requests; any host not in the file **/usr/nserve/auth.info/***domain*/passwd for the **domain** they belong to will not be allowed to mount resources from your host. If **-v** is not specified, hosts named in *domain*/passwd will be verified, and other hosts will be allowed to connect without verification.

-p *primary_addr*
Indicates the primary domain name server for your domain. *primary_addr* must be the network address of the primary name server for your domain. If the **-p** option is not specified, the address of the domain name server is taken from the **rfmaster** file. [See **rfmaster**(4) for a description of the valid address syntax.]

If the host password has not been set, **rfstart** will prompt for a password; the password prompting process must match the password entered for your machine at the primary domain name server [see **rfadmin**(1M)]. If you remove the **loc.passwd** file or change domains, you will also have to reenter the password.

Also, when **rfstart** is run on a domain name server, entries in the **rfmaster**(4) file are syntactically validated.

This command is restricted to the super-user.

ERRORS

If syntax errors are found in validating the **rfmaster**(4) file, a warning describing each error will be sent to standard error.

If (1) the shared resource environment is already running, (2) there is no communications network, (3) the domain name server cannot be found, (4) the domain name server does not recognize the machine, or (5) the command is run without super-user privileges, an error message will be sent to standard error.

Remote file sharing will not start if the host password in /usr/nserve/loc.passwd is corrupted. If you suspect this has happened, remove the file and run **rfstart** again to reenter your password.

NOTE: **rfstart** will NOT fail if your host password does not match the password on the domain name server. You will simply receive a warning message. However, if you try to mount a resource from the primary or any other host that validates your password, the mount will fail if your password does not match the one that host has listed for your machine.

FILES

/usr/nserve/rfmaster
/usr/nserve/loc.passwd

SEE ALSO

adv(1M), dname(1M), mount(1M), rfadmin(1M), rfstop(1M), unadv(1M).
rfmaster(4) in the *Programmer's Reference Manual.*

NAME

rfuadmin – Remote File Sharing notification shell script

SYNOPSIS

rfuadmin *message remote_resource* [*seconds*]

DESCRIPTION

The **rfuadmin** administrative shell script responds to unexpected Remote File Sharing events, such as broken network connections and forced unmounts, picked up by the **rfudaemon** process. This command is not intended to be run directly from the shell.

The response to messages received by **rfudaemon** can be tailored to suit the particular system by editing the **rfuadmin** script. The following paragraphs describe the arguments passed to **rfuadmin** and the responses.

disconnect *remote_resource*

A link to a remote resource has been cut. **rfudaemon** executes **rfuadmin,** passing it the message **disconnect** and the name of the disconnected resource. **rfuadmin** sends this message to all terminals using **wall**(1):

Remote_resource **has been disconnected from the system.**

Then it executes **fuser**(1M) to kill all processes using the resource, unmounts the resource [**umount**(1M)] to clean up the kernel, and starts **rmount** to try to remount the resource.

fumount *remote_resource*

A remote server machine has forced an unmount of a resource a local machine has mounted. The processing is similar to processing for a disconnect.

fuwarn *remote_resource seconds*

This message notifies **rfuadmin** that a resource is about to be unmounted. **rfudaemon** sends this script the **fuwarn** message, the resource name, and the number of seconds in which the forced unmount will occur. **rfuadmin** sends this message to all terminals:

Remote_resource **is being removed from the system in # seconds.**

SEE ALSO

fumount(1M), rmount(1M), rfstart(1M), rfudaemon(1M), wall(1).

BUGS

The console must be on when Remote File Sharing is running. If it's not, **rfuadmin** will hang when it tries to write to the console (**wall**) and recovery from disconected resources will not complete.

NAME

rfudaemon – Remote File Sharing daemon process

SYNOPSIS

rfudaemon

DESCRIPTION

The **rfudaemon** command is started automatically by **rfstart**(1M) and runs as a daemon process as long as Remote File Sharing is active. Its function is to listen for unexpected events, such as broken network connections and forced unmounts, and to execute appropriate administrative procedures.

When such an event occurs, **rfudaemon** executes the administrative shell script **rfuadmin**, with arguments that identify the event. This command is not intended to be run from the shell. Here are the events:

DISCONNECT
A link to a remote resource has been cut. **rfudaemon** executes **rfuadmin** with two arguments: **disconnect** and the name of the disconnected resource.

FUMOUNT
A remote server machine has forced an unmount of a resource a local machine has mounted. **rfudaemon** executes **rfuadmin** with two arguments: **fumount** and the name of the disconnected resource.

GETUMSG
A remote user-level program has sent a message to the local **rfudaemon**. Currently the only message sent is *fuwarn,* which notifies **rfuadmin** that a resource is about to be unmounted. It sends **rfuadmin** the *fuwarn,* the resource name, and the number of seconds in which the forced unmount will occur.

LASTUMSG
The local machine wants to stop the **rfudaemon** [**rfstop**(1M)]. This causes **rfudaemon** to exit.

SEE ALSO

rfstart(1M), rfuadmin(1M).

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [**–f**] [**–i**] file ...

rm –r [**–f**] [**–i**] dirname ... [file ...]

rmdir [**–p**] [**–s**] dirname ...

DESCRIPTION

The *rm* command 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. If a directory is writable and has the sticky bit set, files within that directory can only be removed if one or more of the following is true (see *unlink(2)*):

> the user owns the file the user owns the directory the file is writable to the user the user is the super-user

If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with **y** (for yes), the file is deleted; otherwise the file remains.

Note that if the standard input is not a terminal, the command will operate as if the **–f** option is in effect.

When the parent directory has the sticky bit set and is writable, *rmdir* removes the named directories, which must be empty, if any of the following is true:

> the parent directory is owned by the user the target directory is owned by the user the target directory is writable to the user the user is the super-user

Three options apply to *rm*:

–f This option causes the removal of all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory was attempted, this option cannot suppress an error message.

–r This option causes the recursive removal of any directories and subdirectories in the argument list. The directory will be emptied of files and removed. Note that the user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are removed without prompting, however, if the **–f** option is used, or if the standard input is not a terminal and the **–i** option is not used.

If the removal of a non-empty, write-protected directory was attempted, the command will always fail (even if the **–f** option is used), resulting in an error message.

-i With this option, confirmation of removal of any write-protected file occurs interactively. It overrides the **-f** option and remains in effect even if the standard input is not a terminal.

Two options apply to *rmdir*:

-p This option allows users to remove the directory *dirname* and its parent directories which become empty. A message is printed on standard output as to whether the whole path is removed or part of the path remains for some reason.

-s This option is used to suppress the message printed on standard error when **-p** is in effect.

DIAGNOSTICS

All messages are generally self-explanatory.
It is forbidden to remove the files "." and ".." in order to avoid the consequences of inadvertently doing something like the following:

```
rm -r .*
```

Both *rm* and *rmdir* return exit codes of 0 if all the specified directories are removed successfully. Otherwise, they return a non-zero exit code.

SEE ALSO

unlink(2), rmdir(2) in the *Programmer's Reference Manual*.

NAME

rmntstat – display mounted resource information

SYNOPSIS

rmntstat [**-h**] [*resource*]

DESCRIPTION

When used with no options, *rmntstat* displays a list of all local Remote File Sharing resources that are remotely mounted, the local path name, and the corresponding clients. *rmntstat* returns the remote mount data regardless of whether a resource is currently advertised; this ensures that resources that have been unadvertised but are still remotely mounted are included in the report. When a *resource* is specified, *rmntstat* displays the remote mount information only for that resource. The **-h** option causes header information to be omitted from the display.

EXIT STATUS

If no local resources are remotely mounted, *rmntstat* will return a successful exit status.

ERRORS

If *resource* (1) does not physically reside on the local machine or (2) is an invalid resource name, an error message will be sent to standard error.

SEE ALSO

mount(1M), fumount(1M), unadv(1M).

NAME

rmount – retry remote resource mounts

SYNOPSIS

/etc/rmount -d[r] *special directory*

DESCRIPTION

The *rmount* command is an administrative shell script that tries to mount remote resource *special* on *directory*. If the remote mount is unsuccessful, *rmount* will wait 60 seconds and try to mount the resource again. This will repeat forever. The RETRIES=0 value in the shell script can be changed to limit the number of times the shell script will try to mount a remote resource. The wait time (TIME=60) can also be changed.

See *mount*(1M) for a description of the options.

FILES

/etc/mnttab	mount table

SEE ALSO

fumount(1M), fuser(1M), mount(1M), rfstart(1M), rfuadmin(1M), setmnt(1M).

mnttab(4) in the *Programmer's Reference Manual.*

NAME

rmountall, rumountall – mount, unmount Remote File Sharing resources

SYNOPSIS

/etc/rmountall [-] " *file-system-table* " [...]
/etc/rumountall [**-k**]

DESCRIPTION

The *rmountall* command is a Remote File Sharing command used to mount remote resources according to a *file-system-table*. (**/etc/fstab** is the recommended *file-system-table*.) The special file name "-" reads from the standard input.

The *rumountall* command causes all mounted remote resources to be unmounted. The **-k** option sends a SIGKILL signal, via **fuser**(1M), to processes that have files open.

These commands may be executed only by the super-user.

The file-system-table format is as follows:

column 1	block special file name of file system
column 2	mount-point directory
column 3	–r if to be mounted read-only; –d if remote resource
column 4	file system type (not use with Remote File Sharing)
column 5+	ignored

White space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

SEE ALSO

fuser(1M), mount(1M), rfstart(1M),
signal(2) in the *Programmer's Reference Manual.*

DIAGNOSTICS

No messages are printed if the remote resources are mounted successfully.

Error and warning messages come from *mount*(1M).

NAME

runacct – run daily accounting

SYNOPSIS

/usr/lib/acct/runacct [mmdd [state]]

DESCRIPTION

runacct is the main daily accounting shell procedure. It is normally initiated via *cron*(1M). *runacct* processes connect, fee, disk, and process accounting files. It also prepares summary files for *prdaily* or billing purposes. *runacct is distributed only to source*

runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into **active**. When an error is detected, a message is written to **/dev/console**, mail [see *mail*(1)] is sent to **root** and **adm**, and *runacct* terminates. *runacct* uses a series of lock files to protect against re-invocation. The files **lock** and **lock1** are used to prevent simultaneous invocation, and **lastdate** is used to prevent more than one invocation per day.

runacct breaks its processing into separate, restartable *states* using **statefile** to remember the last *state* completed. It accomplishes this by writing the *state* name into **statefile**. *runacct* then looks in **statefile** to see what it has done and to determine what to process next. *States* are executed in the following order:

SETUP	Move active accounting files into working files.
WTMPFIX	Verify integrity of **wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in **ctmp.h** format.
CONNECT2	Convert **ctmp.h** records into **tacct.h** format.
PROCESS	Convert process accounting records into **tacct.h** format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of *chargefee* into **tacct.h** format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.
MERGETACCT	Merge the daily total accounting records in **daytacct** with the summary total accounting records in **/usr/adm/acct/sum/tacct**.
CMS	Produce command summaries.
USEREXIT	Any installation-dependent accounting programs can be included here.
CLEANUP	Cleanup temporary files and exit.

To restart *runacct* after a failure, first check the **active** file for diagnostics, then fix up any corrupted data files such as **pacct** or **wtmp**. The **lock** files and **lastdate** file must be removed before *runacct* can be restarted. The argument *mmdd* is necessary if *runacct* is being restarted, and specifies the month and day for which *runacct* will rerun the accounting. Entry point for processing is based on the contents of **statefile**; to override this, include the desired *state* on the command line to designate where processing should begin.

EXAMPLES

To start *runacct*.

```
nohup runacct 2> /usr/adm/acct/nite/fd2log &
```

To restart *runacct*.

```
nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &
```

To restart *runacct* at a specific *state*.

```
nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log &
```

FILES

/etc/wtmp
/usr/adm/pacct*
/usr/src/cmd/acct/tacct.h
/usr/src/cmd/acct/ctmp.h
/usr/adm/acct/nite/active
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/statefile
/usr/adm/acct/nite/ptacct*.*mmdd*

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), mail(1).
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

BUGS

Normally it is not a good idea to restart *runacct* in the **SETUP** *state*. Run **SETUP** manually and restart via:

runacct *mmdd* **WTMPFIX**

If *runacct* failed in the **PROCESS** *state*, remove the last **ptacct** file because it will not be complete.

NAME

sag – system activity graph

SYNOPSIS

sag [options]

DESCRIPTION

The *sag* command 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. The *sag* command invokes *sar* and finds the desired data by string-matching the data column header (run *sar* to see what is available). These *options* are passed through 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/sa***dd*.

Other *options*:

-T *term*	Produce output suitable for terminal *term*. See *tplot*(1G) for known terminals. 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/sa*dd*	daily data file for day *dd*.

SEE ALSO

sar(1), tplot(1G)

NAME

sar – system activity reporter

SYNOPSIS

sar [**-ubdycwaqvmprDSAC**] [**-o** file] t [n]

sar [**-ubdycwaqvmprDSAC**] [**-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, where *t* should be 5 or greater. 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 the **-f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* 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. When used with **-D**, %sys is split into percent of time servicing requests from remote machines (%sys remote) and all other system time (%sys local).

-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;
%rcache, %wcache – cache hit ratios, i. e., (1–bread/lread) as a percentage;
pread/s, pwrit/s – transfers via raw (physical) device mechanism. When used with **-D**, buffer caching is reported for locally-mounted remote resources.

-d Report activity for each block device, e. g., disk or tape drive. When data is displayed, the device specification *dsk-* is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is:
%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;
avwait, 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. When used with **-D**, the system calls are split into incoming, outgoing, and strictly local 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 and swapouts (including initial loading of some programs);
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 process, i-node, file tables:
text-sz, proc-sz, inod-sz, file-sz, lock-sz – entries/size for each table, evaluated once at sampling point;
ov – overflows that occur between sampling points for each table.

-m Report message and semaphore activities:
msg/s, sema/s – primitives per second.

-p Report paging activities:
vflt/s – address translation page faults (valid page not in memory);
pflt/s – page faults from protection errors (illegal access to page) or "copy-on-writes";
pgfil/s – vflt/s satisfied by page-in from file system;
rclm/s – valid pages reclaimed for free list.

-r Report unused memory pages and disk blocks:
freemem – average pages available to user processes;
freeswap – disk blocks available for process swapping.

-D Report Remote File Sharing activity:
When used in combination with **-u**, **-b** or **-c**, it causes **sar** to produce the remote file sharing version of the corresponding report. **-Du** is assumed when only **-D** is specified.

-S Report server and request queue status:
Average number of Remote File Sharing servers on the system (serv/lo-hi), % of time receive descriptors are on the request queue (request %busy), average number of receive descriptors waiting for service when queue is occupied (request avg lgth), % of time there are idle servers (server %avail), average number of idle servers when idle ones exist (server avg avail).

-A Report all data. Equivalent to **-udqbwcayvmprSDC**.

-C Report Remote File Sharing buffer caching overhead:
snd-inv/s - number of invalidation messages per second sent by your machine as a server.
snd-msg/s - total outgoing RFS messages sent per second.
rcv-inv/s - number of invalidation messages received from the remote server.
rcv-msg/s - total number of incoming RFS messages received per second.
dis-bread/s - number of buffer reads that would be eligible for caching if caching were not turned off. (Indicates the penalty of running uncached.)
blk-inv/s - number of buffers removed from the client cache.

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/sa*dd*	daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G), sar(1M).

NAME

sar: sa1, sa2, sadc – system activity report package

SYNOPSIS

/usr/lib/sa/sadc [t n] [ofile]

/usr/lib/sa/sa1 [t n]

/usr/lib/sa/sa2 [**-ubdycwaqvmprDSAC**] [**-s** time] [**-e** time] [**-i** sec]

DESCRIPTION

System activity data can be accessed at the special request of a user [see *sar*(1)] and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-call activity, file-access, queue activity, inter-process communications, paging and Remote File Sharing.

sadc and shell procedures, *sa1* and *sa2*, are used to sample, save, and process this data.

sadc, the data collector, samples system data *n* times, with an interval of *t* seconds between samples and writes in binary format to *ofile* or to standard output. If *t* and *n* are omitted, a special record is written. This facility is used at system boot time, when booting to a multiuser state, to mark the time at which the counters restart from zero. For example, the **/etc/init.d/perf** file writes the restart mark to the daily data by the command entry:

```
su sys –c "/usr/lib/sa/sadc /usr/adm/sa/sadate +%d"
```

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file **/usr/adm/sa/sa***dd* where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The entries in **/usr/spool/cron/crontabs/sys** [see *cron*(1M)]:

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8–17 * * 1–5 /usr/lib/sa/sa1
```

will produce records every 20 minutes during working hours and hourly otherwise.

The shell script *sa2*, a variant of *sar*(1), writes a daily report in file **/usr/adm/sa/sar***dd*. The options are explained in *sar*(1). The **/usr/spool/cron/crontabs/sys** entry:

```
5 18 * * 1–5 /usr/lib/sa/sa2 –s 8:00 –e 18:01 –i 1200 –A
```

will report important activities hourly during the working day.

The structure of the binary daily data file is:

```
struct sa {
        struct sysinfo si;  /* see /usr/include/sys/sysinfo.h */
        struct minfo mi;  /* defined in sys/sysinfo.h */
        struck dinfo di;  /* RFS info defined in sys/sysinfo.h */
        int minserve, maxserve;   /* RFS server low and high water marks */
        int  szinode;       /* current size of inode table  */
        int  szfile;        /* current size of file table  */
        int  szproc;        /* current size of proc table  */
        int   szlckf;       /* current size of file record header table */
        int   szlckr;       /* current size of file record lock table */
        int  mszinode;      /* size of inode table  */
        int  mszfile;       /* size of file table  */
        int  mszproc;       /* size of proc table  */
        int   mszlckf;      /* maximum size of file record header table */
        int   mszlckr;      /* maximum size of file record lock table */
        long  inodeovf;     /* cumulative overflows of inode table  */
        long  fileovf;      /* cumulative overflows of file table  */
        long  procovf;      /* cumulative overflows of proc table  */
        time_t  ts;         /* time stamp, seconds  */
        long  devio[NDEVS][4];     /* device unit information  */
#define IO_OPS          0          /* cumulative I/O requests  */
#define IO_BCNT         1          /* cumulative blocks transferred */
#define IO_ACT          2          /* cumulative drive busy time in ticks  */
#define IO_RESP         3          /* cumulative I/O resp time in ticks */
};
```

FILES

/usr/adm/sa/sa*dd*	daily data file
/usr/adm/sa/sar*dd*	daily report file
/tmp/sa.adrfl	address file

SEE ALSO

cron(1M), sag(1G), sar(1), timex(1).

NAME

sdiff – side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

The *sdiff* command 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

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\\xdefx**, 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-printable characters are displayed in octal notation 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.

(1) **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:

n n = 1 - 512. Substitute for just the n-th occurrence of the *regular expression.*

g Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Print the pattern space if a replacement was made.

w *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.

(0) **#** If a **#** appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the **#** is an 'n', then the default output will be suppressed. The rest of the line after **#**n is also ignored. A script file must contain at least one non-comment line.

SEE ALSO

awk(1), ed(1), grep(1).

NAME

setmnt – establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

The *setmnt* command creates the **/etc/mnttab** table, which is needed for both the *mount*(1M) and *umount* commands. The *setmnt* command reads standard input and creates a *mnttab* entry for each line. Input lines have the format:

```
filesys node
```

where *filesys* is the name of the file system's *special file* (e.g., **/dev/dsk/1s1**) and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the mount table entry.

FILES

/etc/mnttab

SEE ALSO

mount(1M).

BUGS

Problems may occur if *filesys* or *node* are longer than 32 characters.
The *setmnt* command silently enforces an upper limit on the maximum number of *mnttab* entries.

NAME

sh, rsh – shell, the standard/restricted command programming language

SYNOPSIS

sh [**-acefhiknrstuvx**] [args]
rsh [**-acefhiknrstuvx**] [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.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *****, **@**, **#**, **?**, **–**, **$**, and **!**.

Commands

A *simple-command* is a sequence of non-blank *words* separated by *blanks.* The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 [see *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 **|**. 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. The exit status of a pipeline is the exit status of the last command.

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") except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list***)**

Execute *list* in a sub-shell.

{*list;*}

list is executed in the current (that is, parent) shell.

name **()** {*list;*}

Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with **#** causes that word and all the following characters up to a new-line to be ignored.

Command Substitution

The shell reads commands from the string between two grave accents (``) and the standard output from these commands may be used as all or part of a word. Trailing new-lines from the standard output are removed.

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" ...\` ...\` ... "), a backslash used to escape a double quote (\\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a new-line character (\new-line), both the backslash and the new-line are removed (see the later section on "Quoting"). In addition, backslashes used to escape dollar signs (\**$**) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \\, \`, ", **new-line**, and **$** are left intact when the command string is read.

Parameter Substitution

The character **$** is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

> *name*=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter*}
: The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or **@**, all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter*:-*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
: If *parameter* is not set or is null, set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}
: If *parameter* is set and is non-null, 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, 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 parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK,
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (**:**) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default "**$** ".

PS2 Secondary prompt string, by default "> ".

IFS Internal field separators, normally **space**, **tab**, and **new-line**.

SHACCT
If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed.

SHELL When the shell is invoked, it scans the environment (see "Environment" below) for this name. If it is found and 'rsh' is the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK**, and **IFS**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments (" " or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Input/Output

A command's input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple-command* or may precede or follow a *command* and are *not* passed on as arguments to the invoked command. Note that parameter and command substitution occurs before *word* or *digit* is used.

<word Use file *word* as standard input (file descriptor 0).

>word Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

>>word Use file *word* as standard output. If the file exists, output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[-]word After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file. If, however, - is appended to <<:

(1) leading tabs are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*),

(2) leading tabs are stripped from the shell input as it is read and before each line is compared with *word*, and

(3) shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file.

If any character of *word* is quoted (see "Quoting," later), no additional processing is done to the shell input. If no characters of *word* are quoted:

(1) parameter and command substitution occurs,

(2) (escaped) \new-line is ignored, and

(3) \\ must be used to quote the characters \\, **$**, and `.

The resulting document becomes the standard input.

<&digit Use the file associated with file descriptor *digit* as standard input; similarly for the standard output using **>&digit**.

<&- The standard input is closed; similarly for the standard output using **>&-**.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1).

For example:

    ... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

    ... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

Using the terminology introduced on the first page, under "Commands," if a *command* is composed of several *simple commands*, redirection will be evaluated for the entire *command* before it is evaluated for each *simple command*. That is, the shell evaluates redirection for the entire *list*, then each *pipeline* within the *list*, then each *command* within each *pipeline*, then each *list* within each *command*.

If a command is followed by **&**, the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

File Name Generation

Before a command is executed, each command *word* is scanned for the characters *****, **?**, and **[**. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, 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 ""!"", 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 backslash (\) or inserting it between a pair of quote marks ('' or

" "). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair **new-line** is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, "'").

Inside a pair of double quote marks (" "), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If **$*** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces (**"$1 $2 ..."**); however, if **$@** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces (**"$1" "$2"** ...). \ quotes the characters \, `, ", and **$**. The pair **new-line** is removed before parameter and command substitution. If a backslash precedes characters other than \, `, ", **$**, and new-line, then the backslash itself is quoted by the shell.

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

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. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd                                and
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **-k** 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 c** and **c**:

```
echo a=b c
set -k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, 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. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command 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, between two colon delimiters anywhere in the path list, or at the end of the path list. If the command name contains a /, 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 is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *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, break *n* levels.

continue [*n*]
: Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified, resume at the *n*-th enclosing loop.

cd [*arg*]
: Change the current directory to *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 / 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*.

echo [*arg* ...]
: Echo arguments. See *echo*(1) for usage and description.

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, 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 *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

getopts
: Use in shell scripts to support command syntax standards [see *intro*(1)]; it parses positional parameters and checks for legal options. See *getopts*(1) for usage and description.

hash [**-r**] [*name* ...]
: For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to

that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* .... See *newgrp*(1M) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]
One line is read from the standard input and, using the internal field separator, **IFS** (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Lines can be continued using **new-line**. Characters other than **new-line** can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *names* are marked *readonly* and the values of the these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**--aefhkntuvx** [*arg* ...]]

-a	Mark variables which are modified or created for export.
-e	Exit immediately if a command exits with a non-zero exit status.
-f	Disable file name generation
-h	Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
-k	All keyword arguments are placed in the environment for a command, not just those that precede the command name.
-n	Read commands but do not execute them.
-t	Exit after reading and executing one command.
-u	Treat unset variables as an error when substituting.
-v	Print shell input lines as they are read.
-x	Print commands and their arguments as they are executed.
--	Do not change any of the 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, 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*] ...
: The command *arg* is 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, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *n* is 0, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
: For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [*n*]
: Impose a size limit of *n* blocks on files written by the shell and its child processes (files of any size may be read). If *n* is omitted, the current limit is printed. You may lower your own ulimit, but only a super-user [see *su*(1M)] can raise a ulimit.

umask [*nnn*]
: The user file-creation mask is set to *nnn* [see *umask*(1)]. If *nnn* is omitted, the current value of the mask is printed.

unset [*name* ...]
: For each *name*, remove the corresponding variable or function. The variables **PATH**, **PS1**, **PS2**, **MAILCHECK**, and **IFS** cannot be unset.

wait [*n*]
: Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be 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 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, commands are read from *string*.
-s	If the **-s** flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.
-i	If the **-i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.
-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.

A restricted shell can be invoked in one of the following ways: (1) *rsh* is the file name part of the last entry in the */etc/passwd* file [see *passwd*(4)]; (2) the environment variable **SHELL** exists and *rsh* is the file name part of its value; (3) the shell is invoked and *rsh* is the file name part of argument 0; (4) the shell is invoked with the **-r** option.

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* [see *profile*(4)] 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 a restricted shell. 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 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), echo(1), env(1), getopts(1), intro(1), login(1), newgrp(1M), pwd(1), test(1), umask(1), wait(1).
dup(2), exec(2), fork(2), pipe(2), profile(4), signal(2), ulimit(2) in the *Programmer's Reference Manual.*

CAVEATS

Words used for file names in input/output redirection are not interpreted for file name generation (see "File Name Generation," above). For example, **cat file1 >a*** will create a file named **a***.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message *cannot fork, too many processes*, try using the *wait*(1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

BUGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For *wait n*, if *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

shl – shell layer manager

SYNOPSIS

shl

DESCRIPTION

The *shl* command allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* is the layer which can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To have the output of a layer blocked when it is not current, the *stty* option **loblk** may be set within the layer.

The *stty* character **swtch** (set to ^Z if NUL) is used to switch control to *shl* from a layer. *shl* has its own prompt, >>>, to help distinguish it from a layer.

A *layer* is a shell which has been bound to a virtual tty device (**/dev/sxt???**). The virtual device can be manipulated like a real tty device using *stty*(1) and *ioctl*(2). Each layer has its own process group id.

Definitions

A *name* is a sequence of characters delimited by a blank, tab, or new-line. Only the first eight characters are significant. The *names* **(1)** through **(7)** cannot be used when creating a layer. They are used by *shl* when no name is supplied. They may be abbreviated to just the digit.

Commands

The following commands may be issued from the *shl* prompt level. Any unique prefix is accepted.

create [*name*]
: Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form *(#)* where # is the last digit of the virtual device bound to the layer. The shell prompt variable **PS1** is set to the name of the layer followed by a space. A maximum of seven layers can be created.

block *name* [*name* ...]
: For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **-loblk** within the layer.

delete *name* [*name* ...]
: For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal [see *signal*(2)].

help (or **?**)
: Print the syntax of the *shl* commands.

layers [**-l**] [*name* ...]
: For each *name*, list the layer name and its process group. The **-l** option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume [*name*]
: Make the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

toggle Resume the layer that was current before the last current layer.

unblock *name* [*name* ...]
: For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **-loblk** within the layer.

quit Exit *shl*. All layers are sent the SIGHUP signal.

name Make the layer referenced by *name* the current layer.

FILES

/dev/sxt???	Virtual tty devices
$SHELL	Variable containing path name of the shell to use (default is /bin/sh).

SEE ALSO

sh(1), stty(1), sxt(7).
ioctl(2), signal(2) in the *Programmer's Reference Manual.*

NAME

shutdown – shut down system, change system state

SYNOPSIS

/etc/shutdown [**-y**] [**-g**grace_period [**-i**init_state]

DESCRIPTION

This command is executed by the super-user to change the state of the machine. By default, it brings the system to a state where only the console has access to the UNIX system. This command can be executed from the console only. This state is traditionally called "single-user".

The command sends a warning message and a final message before it starts actual shutdown activities. By default, the command asks for confirmation before it starts shutting down daemons and killing processes. The options are used as follows:

-y pre-answers the confirmation question so the command can be run without user intervention. A default of 60 seconds is allowed between the warning message and the final message. Another 60 seconds is allowed between the final message and the confirmation.

-g*grace_period*
allows the super-user to change the number of seconds from the 60-second default.

-i*init_state*
specifies the state that *init*(1M) is to be put in following the warnings, if any. By default, system state "**s**" is used (the same as states "**1**" and "**S**").

Other recommended system state definitions are:

state 0
Shut the machine down so it is safe to remove the power. Have the machine remove power if it can. The */etc/rc0* procedure is called to do this work.

state 1, s, S
Bring the machine to the state traditionally called single-user. The */etc/rc0* procedure is called to do this work. (Though **s** and **1** are both used to go to single user state, **s** only kills processes spawned by **init** and does not unmount file systems. State **1** unmounts everything except root and kills all user processes, except those that relate to the console.)

state 6
Stop the UNIX system and reboot to the state defined by the *initdefault* entry in **/etc/inittab**.

SEE ALSO

init(1M), rc0(1M), rc2(1M).
inittab(4) in the *Programmer's Reference Manual.*

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

The *sleep* command 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) in the *Programmer's Reference Manual.*

NAME

sort – sort and/or merge files

SYNOPSIS

sort **[-cmu]** [**-o**output] [-ykmem] [-zrecsz] **[-dfiMnr]** [-**bt**x]
[+pos1 [-pos2]] [files]

DESCRIPTION

sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if **-** is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine-collating sequence.

The following options alter the default behavior:

-c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

-m Merge only, the input files are already sorted.

-u Unique: suppress all but one in each set of lines having equal keys.

-o*output*
The argument given 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. There may be optional blanks between **-o** and *output.*

-y*kmem*
The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem, sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, **-y0** is guaranteed to start with minimum memory. By convention, **-y** (with no argument) starts with maximum memory.

-z*recsz*
The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **-c** or **-m** options, a popular system default size will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

-d "Dictionary" order: only letters, digits, and blanks (spaces and tabs) are significant in comparisons.

-f Fold lower-case letters into upper case.

-i Ignore non-printable characters.

-M Compare as months. The first three non-blank characters of the field are folded to upper case and compared. For example, in English the sorting order is "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **-M** option implies the **-b** option (see the following text).

-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. The **-n** option implies the **-b** option (see the following text). Note that the **-b** option is only effective when restricted sort key specifications are in effect.

-r Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described in the following text), the specified ordering options override all global ordering options for that key.

The notation *+pos1 -pos2* restricts a sort key to one beginning at *pos1* and ending just before *pos2*. The characters at position *pos1* and just before *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *-pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

-b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **-b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *-pos2* argument (see below).

-t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (for example, *xx* delimits an empty field).

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfinr**. A starting position specified by *+m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect, *n* is counted from the first non-blank in the *m*+1st field; *+m*.0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by *-m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m-th* field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect, *n* is counted from the last leading blank in the *m*+1st field; *-m*.1**b** refers to the first non-blank in the *m*+1st field.

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.

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 –2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort –r –o outfile +1.0 –1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

```
sort –r +1.0b –1.1b infile1 infile2
```

Print the password file [*passwd*(4)] sorted by the numeric user ID (the third colon-separated field):

```
sort –t: +2n –3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **–um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort –um +2 –3 infile
```

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

WARNINGS

Comments and exits with non-zero status for various trouble conditions (for example, when input lines are too long), and for disorder discovered under the **–c** option. When the last line of an input file is missing, a **new-line** character, *sort* appends one, prints a warning message, and continues.

sort does not guarantee preservation of relative line ordering on equal keys.

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

The *spell* command 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.

The *spell* command 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*.

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), sed(1), sort(1), tee(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*.

NAME

spline – interpolate smooth curve

SYNOPSIS

spline [options]

DESCRIPTION

The *spline* command 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 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

The *split* command 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

strace – print STREAMS trace messages

SYNOPSIS

strace [mid sid level] ...

DESCRIPTION

The *strace* command without arguments writes all STREAMS event trace messages from all drivers and modules to its standard output. These messages are obtained from the STREAMS log driver [*log*(7)]. If arguments are provided they must be in triplets of the form *mid, sid, level,* where *mid* is a STREAMS module id number, *sid* is a sub-id number, and *level* is a tracing priority level. Each triplet indicates that tracing messages are to be received from the given module/driver, sub-id (usually indicating minor device), and priority level equal to or less than the given level. The token *all* may be used for any member to indicate no restriction for that attribute.

The format of each trace message output is:

<seq> <time> <ticks> <level> <flags> <mid> <sid> <text>

<seq>	trace sequence number
<time>	time of message in hh:mm:ss
<ticks>	time of message in machine ticks since boot
<level>	tracing priority level
<flags>	E : message is also in the error log F : indicates a fatal error N : mail was sent to the system administrator
<mid>	module id number of source
<sid>	sub-id number of source
<text>	formatted text of the trace message

Once initiated, *strace* will continue to execute until terminated by the user.

EXAMPLES

Output all trace messages from the module or driver whose module id is 41:

strace 41 all all

Output those trace messages from driver/module id 41 with sub-ids 0, 1, or 2:

strace 41 0 1 41 1 1 41 2 0

Messages from sub-ids 0 and 1 must have a tracing level less than or equal to 1. Those from sub-id 2 must have a tracing level of 0.

CAVEATS

Due to performance considerations, only one *strace* process is permitted to open the STREAMS log driver at a time. The log driver has a list of the triplets specified in the command invocation, and compares each potential trace message against this list to decide if it should be formatted and sent up to the *strace* process. Hence, long lists of triplets will have a greater impact on overall STREAMS performance. Running *strace* will have the most impact on the timing of the modules and drivers generating the trace messages that are sent to the *strace* process. If trace messages are generated faster than the *strace* process can handle them, then some of the messages will be lost. This last case can be determined by examining the sequence numbers on the trace messages output.

SEE ALSO

log(7).
STREAMS Programmer's Guide.

NAME

strclean – STREAMS error logger cleanup program

SYNOPSIS

strclean [**-d** logdir] [**-a** age]

DESCRIPTION

The *strclean* command is used to clean up the STREAMS error logger directory on a regular basis [for example, by using *cron*(1M)]. By default, all files with names matching **error.*** in **/usr/adm/streams** that have not been modified in the last 3 days are removed. A directory other than **/usr/adm/streams** can be specified using the **-d** option. The maximum age in days for a log file can be changed using the **-a** option.

EXAMPLE

```
strclean -d /usr/adm/streams -a 3
```

has the same result as running strclean with no arguments.

NOTES

The *strclean* command is typically run from *cron*(1M) on a daily or weekly basis.

FILES

/usr/adm/streams/error.*

SEE ALSO

cron(1M), strerr(1M).
STREAMS Programmer's Guide.

NAME

strerr – STREAMS error logger daemon

SYNOPSIS

strerr

DESCRIPTION

The *strerr* routine receives error log messages from the STREAMS log driver [*log*(7)] and appends them to a log file. The error log files produced reside in the directory **/usr/adm/streams**, and are named **error.***mm-dd*, where *mm* is the month and *dd* is the day of the messages contained in each log file.

The format of an error log message is:

<seq> <time> <ticks> <flags> <mid> <sid> <text>

<seq>	error sequence number
<time>	time of message in hh:mm:ss
<ticks>	time of message in machine ticks since boot priority level
<flags>	T : the message was also sent to a tracing process F : indicates a fatal error N : send mail to the system administrator
<mid>	module id number of source
<sid>	sub-id number of source
<text>	formatted text of the error message

Messages that appear in the error log are intended to report exceptional conditions that require the attention of the system administrator. Those messages which indicate the total failure of a STREAMS driver or module should have the F flag set. Those messages requiring the immediate attention of the administrator will have the N flag set, which causes the error logger to send the message to the system administrator via *mail*(1). The priority level usually has no meaning in the error log but will have meaning if the message is also sent to a tracer process.

Once initiated, *strerr* will continue to execute until terminated by the user. Commonly, *strerr* would be executed asynchronously.

CAVEATS

Only one *strerr* process at a time is permitted to open the STREAMS log driver.

If a module or driver is generating a large number of error messages, running the error logger will cause a degradation in STREAMS performance. If a large burst of messages are generated in a short time, the log driver may not be able to deliver some of the messages. This situation is indicated by gaps in the sequence numbering of the messages in the log files.

FILES

/usr/adm/streams/error.*mm-dd*

SEE ALSO

log(7).
STREAMS Programmer's Guide.

NAME

stty – set the options for a terminal

SYNOPSIS

stty [**-a**] [**-g**] [options]

DESCRIPTION

The *stty* command sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

In this report, if a character is preceded by a caret (^), then the value of that option is the corresponding CTRL character (e.g., "^**h**" is **CTRL-h ;** in this case, recall that **CTRL-h** is the same as the "back-space" key.) The sequence "^'" means that an option has a null value. For example, normally **stty -a** will report that the value of **swtch** is "^'"; however, if **shl (1)** or **layers (1)** has been invoked, **stty -a** will have the value "^**z**".

-a	reports all of the option settings;
-g	reports current settings in a form that can be used as an argument to another *stty* command.

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.
110 300 600 1200 1800 2400 4800 9600 19200 38400	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) Dataphone data set 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)	n assume a line without (with) modem control.
loblk (-loblk)	block (do not block) output from a non-current layer.

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.
iuclc (-iuclc)	map (do not map) uppercase alphabetics to lowercase 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) lowercase alphabetics to upper case on output.
onlcr (-onlcr)	map (do not map) NL to CR-NL on output.
ocrnl (-ocrnl)	map (do not map) CR to NL on output.
onocr (-onocr)	do not (do) output CRs at column zero.
onlret (-onlret)	on the terminal NL performs (does not perform) the CR function.
ofill (-ofill)	use fill characters (use timing) for delays.
ofdel (-ofdel)	fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	select style of delay for carriage returns [see *termio*(7)].
nl0 nl1	select style of delay for line-feeds [see *termio*(7)].
tab0 tab1 tab2 tab3	select style of delay for horizontal tabs [see *termio*(7)].
bs0 bs1	select style of delay for backspaces [see *termio*(7)].
ff0 ff1	select style of delay for form-feeds [see *termio*(7)].
vt0 vt1	select style of delay for vertical tabs [see *termio*(7)].

Local Modes

isig (-isig)	enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.
icanon (-icanon)	enable (disable) canonical input (ERASE and KILL processing).
xcase (-xcase)	canonical (unprocessed) uppercase/lowercase presentation.
echo (-echo)	echo back (do not echo back) every character typed.
echoe (-echoe)	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 *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (-echok)	echo (do not echo) NL after KILL character.
lfkc (-lfkc)	the same as **echok (-echok)**; obsolete.
echonl (-echonl)	echo (do not echo) NL.
noflsh (-noflsh)	disable (enable) flush after INTR, QUIT, or SWTCH.

stwrap (-stwrap)	disable (enable) truncation of lines longer than 79 characters on a synchronous line.
stflush (-stflush)	enable (disable) flush on a synchronous line after every *write*(2).
stappl (-stappl)	use application mode (use line mode) on a synchronous line.

Control Assignments

control-character c	set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **swtch**, **eof**, **eol**, **ctab**, **min**, or **time** [**ctab** is used with **-stappl**; **min** and **time** are used with **-icanon**; see *termio*(7)]. If *c* 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.
line *i*	set line discipline to *i* (0 < *i* < 127).

Combination Modes

evenp or **parity**	enable **parenb** and **cs7**.
oddp	enable **parenb**, **cs7**, and **parodd**.
-parity, **-evenp**, or **-oddp**	disable **parenb**, and set **cs8**.
raw (**-raw** or **cooked**)	enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).
nl (-nl)	unset (set) **icrnl**, **onlcr**. In addition **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (-lcase)	set (unset) **xcase**, **iuclc**, 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), termio(7).
ioctl(2) in the *Programmer's Reference Manual.*

NAME

su – become super-user or another user

SYNOPSIS

su [-] [name [arg ...]] **-c -r**

DESCRIPTION

The *su* command 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 **root**). If the password is correct, *su* will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry [see *passwd*(4)], or **/bin/sh** if none is specified [see *sh*(1)]. To restore normal user ID privileges, type an **EOF** (*cntrl-d*) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like *sh*(1), an *arg* of the form **-c** *string* executes *string* via the shell and an arg of **-r** will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(1). If the first argument to *su* is a -, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is -, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) 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 if the optional program used as the shell is **/bin/sh**, the user's **.profile** can check *arg0* for **-sh** or **-su** to determine if it was invoked by *login*(1) or *su*(1M), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *-program* by both *login*(1) and *su*(1M).

All attempts to become another user using *su* are logged in the log file **/usr/adm/sulog**.

EXAMPLES

To become user **bin** while retaining your previously exported environment, execute:

```
/bin/su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
/bin/su – bin
```

To execute *command* with the temporary environment and permissions of user **bin**, type:

```
/bin/su – bin –c "command args"
```

FILES

/etc/passwd	system's password file

/etc/profile	system's profile
$HOME/.profile	user's profile
/usr/adm/sulog	log file

SEE ALSO

env(1), login(1), sh(1).
passwd(4), profile(4), environ(5) in the *Programmer's Reference Manual.*

NAME

sum – print checksum and block count of a file

SYNOPSIS

sum [**-r**] file

DESCRIPTION

The *sum* command 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

swap – swap administrative interface

SYNOPSIS

/etc/swap -a swapdev swaplow swaplen
/etc/swap -d swapdev swaplow
/etc/swap -l

DESCRIPTION

The *swap* command provides a method of adding, deleting, and monitoring the system swap areas used by the memory manager. The following options are recognized:

-a Add the specified swap area. *swapdev* is the name of the block special device, e.g., **/dev/dsk/1s0**. *swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. *swaplen* is the length of the swap area in 512-byte blocks. This option can only be used by the super-user. Swap areas are normally added by the system start-up routine **/etc/rc** when going into multiuser mode.

-d Delete the specified swap area. *swapdev* is the name of a block special device, e.g., **/dev/dsk/1s0**. *swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. This option can only be used by the super-user.

-l List the status of all the swap areas. The output has four columns:

DEV	The *swapdev* special file for the swap area if one can be found in the **/dev/dsk** or **/dev** directories, and its major/minor device number in decimal.
LOW	The *swaplow* value for the area in 512-byte blocks.
LEN	The *swaplen* value for the area in 512-byte blocks.
FREE	The number of free 512-byte blocks in the area.

WARNINGS

No check is done to see if a swap area being added overlaps with an existing swap area or file system.

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

The *sync* command 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.

NOTE

If you have done a write to a file on a remote machine in a Remote File Sharing environment, you cannot use *sync* to force buffers to be written out to disk on the remote machine. *sync* will only write local buffers to local disks.

SEE ALSO

sync(2) in the *Programmer's Reference Manual.*

NAME

sysdef – output system definition

SYNOPSIS

/etc/sysdef [system_namelist [master.d]]

DESCRIPTION

The *sysdef* command outputs the current system definition in tabular form. It lists all hardware devices, their local bus addresses, and unit count, as well as pseudo devices, system devices, loadable modules and the values of all tunable parameters. It generates the output by analyzing the named operating system file (*system_namelist*) and extracting the configuration information from the name list itself.

FILES

/unix	default operating system file (where the system namelist is)
/etc/master.d/*	default directory containing master files

SEE ALSO

master(4), nlist(3C) in the *Programmer's Reference Manual.*

DIAGNOSTICS

internal name list overflow

if the master table contains more than an internally specified number of entries for use by *nlist*(3C).

NAME

tabs – set tabs on a terminal

SYNOPSIS

tabs [tabspec] [-Ttype] [+**mn**]

DESCRIPTION

The *tabs* command sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

tabspec Four types of tab specification are accepted for *tabspec*. They are described below: canned (*-code*), repetitive (*-n*), arbitrary (*n1,n2,...*), and file (*--file*). If no *tabspec* is given, the default value is **-8**, i.e., UNIX system "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 Use one of the codes listed below to select a *canned* set of 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 [see *fspec*(4)]: **<: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 [see *fspec*(4)]: **<: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

-n A *repetitive* specification requests tabs at columns 1+*n*, 1+2**n*, etc. Of particular importance is the value **8**: this represents the UNIX system "standard" tab setting, and is the most likely tab setting to be found at a terminal. 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 formats **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 [see *fspec*(4)]. 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 also may be used; if a given flag occurs more than once, the last value given takes effect:

-T*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* uses the value of the environment variable **TERM**. If **TERM** is not defined in the *environment* [see *environ*(5)], *tabs* tries a sequence that will work for many terminals.

+m*n* 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.

EXAMPLES

tabs -a example using *-code* (*canned* specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

tabs -8 example of using *-n* (*repetitive* specification), where *n* is **8**, causes tabs to be set every eighth position:
1+(1*8), 1+(2*8), ... which evaluate to columns 9, 17, ...

tabs 1,8,36 example of using *n1,n2,...* (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

tabs --$HOME/fspec.list/att4425
: example of using *--file* (*file* specification) to indicate that tabs should be set according to the first line of *$HOME/fspec.list/att4425* [see *fspec*(4)].

DIAGNOSTICS

illegal tabs	when arbitrary tabs are ordered incorrectly
illegal increment	when a zero or missing increment is found in an arbitrary specification
unknown tab code	when a *canned* code cannot be found
can't open	if *--file* option used, and file can't be opened
file indirection	if *--file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted

SEE ALSO

newform(1), pr(1), tput(1).
fspec(4), terminfo(4), environ(5), term(5) in the *Programmer's Reference Manual.*

NOTE

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

WARNING

The *tabspec* used with the *tabs* command is different from the one used with the *newform*(1) command. For example, **tabs -8** sets every eighth position; whereas **newform -i-8** indicates that tabs are set every eighth position.

NAME

tail – deliver the last part of a file

SYNOPSIS

tail [±[number][**lbc**[**f**]]] [file]

DESCRIPTION

The *tail* command 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(1M).

BUGS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

WARNING

The *tail* command will only tail the last 4096 bytes of a file regardless of its line count.

NAME

tapecntl – tape control for QIC-24/QIC-02 tape device

SYNOPSIS

tapecntl [**-etrw**] [**-p arg**]

DESCRIPTION

Tapecntl will send the optioned commands to the tape device driver sub-device **/dev/rmt/c0s0** for all commands except "position," which will use sub-device **/dev/rmt/c0s0n** using the ioctl command function. Sub-device **/dev/rmt/c0s0** provides a rewind on close capability, while **/dev/rmt/c0s0n** allows for closing of the device without rewind. Error messages will be written to standard error.

-e	erase tape
-t	retension tape
-r	reset tape device
-w	rewind tape
-p[n]	position tape to "end of file" mark – n

Erasing the tape causes the erase bar to be activated while moving the tape from end to end, causing all data tracks to be erased in a single pass over the tape. Retensioning the tape causes the tape to be moved from end to end, thereby repacking the tape with the proper tension across its length. Reset of the tape device initializes the tape controller registers and positions the tape at the beginning of the tape mark (BOT). Rewinding the tape will move the tape to the BOT. Positioning the tape command requires an integer argument. Positioning the tape will move the tape forward relative to its current position to the end of the specified file mark. The positioning option used with an argument of zero will be ignored. Illegal or out-of-range value arguments to the positioning command will leave the tape positioned at the end of the last valid file mark. Options may be used individually or strung together with selected options being executed sequentially from left to right in the command line.

FILES

/usr/lib/tape/tapecntl
/dev/rmt/c0s0n
/dev/rmt/c0s0

NOTES

Exit codes and their meanings are as follows:

exit (1)	device function could not initiate properly due to misconnected cables or poorly inserted tape cartridge.
exit (2)	device function failed to complete properly due to unrecoverable error condition, either in the command setup or due to mechanical failure.
exit (3)	device function failed due to the cartridge being write protected or to the lack of written data on the tape.
exit (4)	device **/dev/rmt/c0s0n** or **/dev/rmt/c0s0** failed to open properly due to already being opened or claimed by another process.

NAME

tar – tape file archiver

SYNOPSIS

/etc/tar -c[vwf] *device block files ...*
/etc/tar -rf[vw] *output_file block files ...*
/etc/tar -t[vf] *device*
/etc/tar -uf[vw] *output_file block files ...*
/etc/tar -x[lmovwf] *device files ...*

DESCRIPTION

The *tar* command saves and restores files on magnetic tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing one function letter (c, r, t, u, or x) and possibly followed by one or more function modifiers (v, w, and f). 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 Replace. The named *files* are written on the end of the archive. The **c** function implies this function. This option applies only to files and will not work with tape.

x Extract. 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. Use the file or directory's relative path when appropriate, or *tar* will not find a match. 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 Table. The names and other information for the specified files are listed each time that they occur on the tape. The listing is similar to the format produced by the **ls -l** command. If no *files* argument is given, all the names on the tape are listed.

u Update. The named *files* are added to the archive if they are not already there, or have been modified since last written on that tape. This key implies the **r** key. This option applies only to files and will not work with tape.

c Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This key implies the **r** key.

The characters below may be used in addition to the letter that selects the desired function. Use them in the order shown in the synopsis. **Note**: the only applicable device information for your computer is as follows:

```
/dev/rmt/c0s0
/dev/rmt/c0s0nr
/dev/rmt/c0s0r
/dev/rmt/c0s0n
```

v Verbose. 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 What. This 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". This is not valid with the **t** key.

f File. This causes *tar* to use the *device* argument as the name of the archive instead of **/dev/rmt/c0s0**. 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 -)
```

l Link. This 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 Modify. This tells *tar* not to restore the modification times. The modification time of the file will be the time of extraction.

o Ownership. This causes extracted files to take on the user and group identifier of the user running the program, rather than those on tape. This is only valid with the **x** key.

FILES

/dev/rmt/*

SEE ALSO

cpio(1), ls(1).

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.

tar doesn't copy empty directories or special files.

NAME

tee – pipe fitting

SYNOPSIS

tee [**-i**] [**-a**] [file] ...

DESCRIPTION

The *tee* command transcribes the standard input to the standard output and makes copies in the *files*. The

-i ignore interrupts;

-a causes the output to be appended to the *files* rather than overwriting them.

NAME

test – condition evaluation command

SYNOPSIS

test expr

[expr **]**

DESCRIPTION

The *test* command evaluates the expression *expr* and, if its value is true, sets a zero (true) exit status; otherwise, a non-zero (false) exit status is set; *test* also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second SYNOPSIS line) must be separate arguments to the *test* command; normally these items are separated by spaces.

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 [*fildes*]	true if the open file whose file descriptor number is *fildes* (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 also that parentheses are meaningful to the shell and, therefore, must be quoted.

SEE ALSO

find(1), sh(1).

WARNING

If you test a file you own (the *-r*, *-w*, or *-x* tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the *group* or *other* bit set for that permission. The correct exit status will be set if you are super-user.

The = and != operators have a higher precedence than the **-r** through **-n** operators, and = and != always expect arguments; therefore, = and != cannot be used with the **-r** through **-n** operators.

If more than one argument follows the **-r** through **-n** operators, only the first argument is examined; the others are ignored, unless a **-a** or a **-o** is the second argument.

NAME

tic – terminfo compiler

SYNOPSIS

tic [**-v**[n]] [**-c**] file

DESCRIPTION

The *tic* command translates a *terminfo*(4) file from the source format into the compiled format. The results are placed in the directory */usr/lib/terminfo*. The compiled format is necessary for use with the library routines described in *curses*(3X).

-vn (verbose) output to standard error trace information showing *tic*'s progress. The optional integer *n* is a number from 1 to 10, inclusive, indicating the desired level of detail of information. If *n* is omitted, the default level is 1. If *n* is specified and greater than 1, the level of detail is increased.

-c only check *file* for errors. Errors in **use=** links are not detected.

file contains one or more *terminfo*(4) terminal descriptions in source format [see *terminfo*(4)]. Each description in the file describes the capabilities of a particular terminal. When a **use=***entry-name* field is discovered in a terminal entry currently being compiled, *tic* reads in the binary from */usr/lib/terminfo* to complete the entry. (Entries created from *file* will be used first. If the environment variable **TERMINFO** is set, that directory is searched instead of */usr/lib/terminfo*.) *tic* duplicates the capabilities in *entry-name* for the current entry, with the exception of those capabilities that explicitly are defined in the current entry.

If the environment variable **TERMINFO** is set, the compiled results are placed there instead of */usr/lib/terminfo*.

FILES

/usr/lib/terminfo/?/* compiled terminal description data base

SEE ALSO

curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual*.
Chapter 10 in the *Programmer's Guide*.

WARNINGS

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Terminal names exceeding 14 characters will be truncated to 14 characters and a warning message will be printed.

When the **-c** option is used, duplicate terminal names will not be diagnosed; however, when **-c** is not used, they will be.

BUGS

To allow existing executables from the previous release of the UNIX system to continue to run with the compiled terminfo entries created by the new terminfo compiler, cancelled capabilities will not be marked as cancelled within the terminfo binary unless the entry name has a '+' within it. (Such terminal names are only used for inclusion within other entries via a **use=** entry. Such names would not be used for real terminal names.)

For example:

```
        4415+nl, kf1@, kf2@, ....

        4415+base, kf1=\EOc, kf2=\EOd, ....

        4415-nl|4415 terminal without keys,
             use=4415+nl, use=4415+base,
```

The above example works as expected; the definitions for the keys do not show up in the *4415-nl* entry. However, if the entry *4415+nl* did not have a plus sign within its name, the cancellations would not be marked within the compiled file and the definitions for the function keys would not be cancelled within *4415-nl*.

DIAGNOSTICS

Most diagnostic messages produced by *tic* during the compilation of the source file are preceded with the approximate line number and the name of the terminal currently being worked on.

mkdir ... returned bad status
: The named directory could not be created.

File does not start with terminal names in column one
: The first thing seen in the file, after comments, must be the list of terminal names.

Token after a *lseek*(2) not NAMES
: Somehow the file being compiled changed during the compilation.

Not enough memory for use_list element
or
Out of memory
: Not enough free memory was available [*malloc*(3C) failed].

Can't open ...
: The named file could not be created.

Error in writing ...
: The named file could not be written to.

Can't link ... to ...
: A link failed.

Error in re-reading compiled file ...
The compiled file could not be read back in.

Premature EOF
: The current entry ended prematurely.

Backspaced off beginning of line
: This error indicates something wrong happened within *tic*.

Unknown Capability - "..."
: The named invalid capability was found within the file.

Wrong type used for capability "..."
: For example, a string capability was given a numeric value.

Unknown token type
: Tokens must be followed by '@' to cancel, ',' for Booleans, '#' for numbers, or '=' for strings.

"...": bad term name
or
Line ...: Illegal terminal name - "..."
Terminal names must start with a letter or digit
: The given name was invalid. Names must not contain white space or slashes, and must begin with a letter or digit.

"...": terminal name too long.
: An extremely long terminal name was found.

"...": terminal name too short.
: A one-letter name was found.

"..." filename too long, truncating to "..."
: The given name was truncated to 14 characters due to UNIX system file name length limitations.

"..." defined in more than one entry. Entry being used is "...".
: An entry was found more than once.

Terminal name "..." synonym for itself
: A name was listed twice in the list of synonyms.

At least one synonym should begin with a letter.
: At least one of the names of the terminal should begin with a letter.

Illegal character - "..."
: The given invalid character was found in the input file.

New-line in middle of terminal name
: The trailing comma was probably left off of the list of names.

Missing comma
: A comma was missing.

Missing numeric value
: The number was missing after a numeric capability.

NULL string value
: The proper way to say that a string capability does not exist is to cancel it.

Very long string found. Missing comma?
: self-explanatory

Unknown option. Usage is:
: An invalid option was entered.

Too many file names. Usage is:
: self-explanatory

"..." nonexistent or permission denied
: The given directory could not be written into.

"..." is not a directory
: self-explanatory

"...": Permission denied
: access denied.

"...": Not a directory
: *tic* wanted to use the given name as a directory, but it already exists as a file

SYSTEM ERROR!! Fork failed!!!
: A *fork*(2) failed.

Error in following up use-links. Either there is a loop in the links or they reference nonexistent terminals. The following is a list of the entries involved:
: A *terminfo*(4) entry with a **use=***name* capability either referenced a nonexistent terminal called *name* or *name* somehow referred back to the given entry.

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 times are printed on standard error.

SEE ALSO

times(2) in the *Programmer's Reference Manual.*

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. The options are as follows:

- **-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).
- **-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. 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

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 the 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

touch – update access and modification times of a file

SYNOPSIS

touch [**-amc**] [mmddhhmm[yy]] files

DESCRIPTION

The *touch* command causes the access and modification times of each argument to be updated. The file name is created if it does not exist. 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) in the *Programmer's Reference Manual.*

NAME

tplot – graphics filters

SYNOPSIS

tplot [**-T**terminal [**-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 *terminal*s 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) in the *Programmer's Reference Manual.*

NAME

tput – initialize a terminal or query terminfo data base

SYNOPSIS

tput [-Ttype] capname [parms ...]

tput [-Ttype] **init**

tput [-Ttype] **reset**

tput [-Ttype] **longname**

DESCRIPTION

The *tput* command uses the *terminfo*(4) data base to make the values of terminal-dependent capabilities and information available to the shell [see *sh*(1)], to initialize or reset the terminal, or return the long name of the requested terminal type. *tput* outputs a string if the attribute (*cap*ability *name*) is of type string, or an integer if the attribute is of type integer. If the attribute is of type Boolean, *tput* simply sets the exit code (**0** for TRUE if the terminal has the capability, **1** for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code [$?, see *sh*(1)] to be sure it is **0**. (See **EXIT CODES** and **DIAGNOSTICS** below.) For a complete list of capabilities and the *capname* associated with each, see *terminfo*(4).

-T*type* indicates the *type* of terminal. Normally this option is unnecessary, because the default is taken from the environment variable **TERM.** If **-T** is specified, then the shell variables **LINES** and **COLUMNS** and the layer size [see *layers*(1)] will not be referenced.

capname indicates the attribute from the *terminfo*(4) data base.

parms If the attribute is a string that takes parameters, the arguments *parms* will be instantiated into the string. An all numeric argument will be passed to the attribute as a number.

init If the *terminfo*(4) data base is present and an entry for the user's terminal exists (see **-T***type*, above), the following will occur: (1) if present, the terminal's initialization strings will be output (**is1**, **is2**, **is3**, **if**, **iprog**), (2) any delays (e.g., new-line) specified in the entry will be set in the tty driver, (3) tabs expansion will be turned on or off according to the specification in the entry, and (4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.

reset Instead of putting out initialization strings, the terminal's reset strings will be output if present (**rs1**, **rs2**, **rs3**, **rf**). If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, **reset** acts identically to **init**.

longname If the *terminfo*(4) data base is present and an entry for the user's terminal exists (see **-T***type* above), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal's description in the *terminfo*(4) data base [see *term*(5)].

EXAMPLES

tput init Initialize the terminal according to the type of terminal in the environmental variable **TERM**. This command should be included in everyone's .profile after the environmental variable **TERM** has been exported, as illustrated on the *profile*(4) manual page.

tput -T5620 reset Reset an AT&T 5620 terminal, overriding the type of terminal in the environmental variable **TERM**.

tput cup 0 0 Send the sequence to move the cursor to row **0**, column **0** (the upper left corner of the screen, usually known as the "home" cursor position).

tput clear Echo the clear-screen sequence for the current terminal.

tput cols Print the number of columns for the current terminal.

tput -T450 cols Print the number of columns for the 450 terminal.

bold='tput smso'

offbold='tput rmso'

Set the shell variables **bold**, to begin stand-out mode sequence, and **offbold**, to end stand-out mode sequence, for the current terminal. This might be followed by a prompt:
echo "${bold}Please type in your name: ${offbold}\c"

tput hc Set exit code to indicate if the current terminal is a hard-copy terminal.

tput cup 23 4 Send the sequence to move the cursor to row 23, column 4.

tput longname Print the long name from the *terminfo*(4) data base for the type of terminal specified in the environmental variable **TERM**.

FILES

/usr/lib/terminfo/?/*	compiled terminal description data base
/usr/include/curses.h	*curses*(3X) header file
/usr/include/term.h	*terminfo*(4) header file
/usr/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see the "Tabs and Initialization" section of *terminfo*(4)

SEE ALSO

stty (1), tabs (1).
profile(4), terminfo(4) in the *Programmer's Reference Manual.*
Chapter 10 of the *Programmer's Guide.*

EXIT CODES

If *capname* is of type Boolean, a value of **0** is set for TRUE and **1** for FALSE.

If *capname* is of type string, a value of **0** is set if the *capname* is defined for this terminal *type* (the value of *capname* is returned on standard output); a value of **1** is set if *capname* is not defined for this terminal *type* (a null value is returned on standard output).

If *capname* is of type integer, a value of **0** is always set, whether or not *capname* is defined for this terminal *type*. To determine if *capname* is defined for this terminal *type*, the user must test the value of standard output. A value of **-1** means that *capname* is not defined for this terminal *type*.

Any other exit code indicates an error; see **DIAGNOSTICS,** below.

DIAGNOSTICS

tput prints the following error messages and sets the corresponding exit codes.

exit code	error message
0	–1 (*capname* is a numeric variable that is not specified in the *terminfo*(4) data base for this terminal type, e.g., **tput -T450 lines** and **tput -T2621 xmc**)
1	no error message is printed, see **EXIT CODES**, above.
2	usage error
3	unknown terminal *type* or no *terminfo*(4) data base
4	unknown *terminfo*(4) capability *capname*

NAME

tr – translate characters

SYNOPSIS

tr [**-cds**] [string1 [string2]]

DESCRIPTION

The *tr* command 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.

EXAMPLE

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 alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for new-line.

```
tr –cs "[A–Z][a–z]" "[\012*]" <file1 >file2
```

SEE ALSO

ed(1), sh(1).
ascii(5) in the *Programmer's Reference Manual*.

BUGS

Will not handle ASCII **NUL** in *string1* or *string2*; always deletes **NUL** from input.

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

tty – get the name of the terminal

SYNOPSIS

tty [**-l**] [**-s**]

DESCRIPTION

The *tty* command prints the path name of the user's terminal.

-l prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line.

-s inhibits printing of the terminal 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

Uutry – try to contact remote system with debugging on

SYNOPSIS

/usr/lib/uucp/Uutry [**-x** debug_level] [**-r**] system_name

DESCRIPTION

Uutry is a shell that is used to invoke *uucico* to call a remote site. Debugging is turned on (default is level 5); **-x** will override that value. The **-r** overrides the retry time in **/usr/spool/uucp/.status**. The debugging output is put in file **/tmp/***system_name.* A tail **-f** of the output is executed. A <DELETE> or <BREAK> will give control back to the terminal while the *uucico* continues to run, putting its output in **/tmp/***system_name.*

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*
/tmp/system_name

SEE ALSO

uucico(1M), uucp(1C), uux(1C).

NAME

uadmin – administrative control

SYNOPSIS

/etc/uadmin cmd fcn

DESCRIPTION

The *uadmin* command provides control for basic administrative functions. This command is tightly coupled to the System Administration procedures and is not intended for general use. It may be invoked only by the super-user.

The arguments *cmd* (command) and *fcn* (function) are converted to integers and passed to the *uadmin* system call.

SEE ALSO

uadmin(2) in the *Programmer's Reference Manual.*

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.

The *umask* command is recognized and executed by the shell.

The *umask* command can be included in the user's **.profile** [see *profile*(4)] and invoked at login to automatically set the user's permissions on files or directories created.

SEE ALSO

chmod(1), sh(1).
chmod(2), creat(2), umask(2), profile(4) in the *Programmer's Reference Manual.*

NAME

unadv – unadvertise a Remote File Sharing resource

SYNOPSIS

unadv *resource*

DESCRIPTION

The *unadv* command unadvertises a Remote File Sharing *resource,* which is the advertised symbolic name of a local directory, by removing it from the advertised information on the domain name server. *unadv* prevents subsequent remote mounts of that resource. It does not affect continued access through existing remote or local mounts.

An administrator at a server can unadvertise only those resources that physically reside on the local machine. A domain administrator can unadvertise any resource in the domain from the primary name server by specifying *resource* name as *domain.resource.* (A domain administrator should only unadvertise another host's resources to clean up the domain advertise table when that host goes down. Unadvertising another host's resource changes the domain advertise table, but not the host advertise table.)

This command is restricted to the super-user.

ERRORS

If *resource* is not found in the advertised information, an error message will be sent to standard error.

SEE ALSO

adv(1M), fumount(1M), nsquery(1M).

NAME

uname – print name of current UNIX system

SYNOPSIS

uname [**-snrvma**]
uname [**-S** system name]

DESCRIPTION

The *uname* command prints the current system name of the UNIX system on the standard output file. It is mainly useful to determine which 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 is the name by which the system is known 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.

On your computer, the system name and the nodename may be changed by specifying a system name argument to the **-S** option. The system name argument is restricted to 8 characters. Only the super-user is allowed this capability.

SEE ALSO

uname(2) in the *Programmer's Reference Manual.*

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [**-udc** [+n] [-n]] [input [output]]

DESCRIPTION

The *uniq* command 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

The *units* command 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
```

The *units* command does only 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

uucheck – check the uucp directories and permissions file

SYNOPSIS

/usr/lib/uucp/uucheck [**-v**] [**-x** debug_level]

DESCRIPTION

The *uucheck* command checks for the presence of the *uucp* system required files and directories. Within the *uucp* makefile, it is executed before the installation takes place. It also checks for some obvious errors in the Permissions file (**/usr/lib/uucp/Permissions**). When executed with the **-v** option, it gives a detailed explanation of how the uucp programs will interpret the Permissions file. The **-x** option is used for debugging. *debug-option* is a single digit in the range 1-9; the higher the value, the greater the detail.

Note that *uucheck* can only be used by the super-user or *uucp*.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuscheds
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

uucico(1M), uucp(1C), uusched(1M), uustat(1C), uux(1C).

BUGS

The program does not check file/directory modes or some errors in the Permissions file such as duplicate login or machine name.

NAME

uucico – file transport program for the uucp system

SYNOPSIS

/usr/lib/uucp/uucico [**-r** role_number] [**-x** debug_level]
[**-i** interface] [**-d** spool_directory] **-s** system_name

DESCRIPTION

The *uucico* command is the file transport program for *uucp* work file transfers. Role numbers for the **-r** are the digit 1 for master mode or 0 for slave mode (default). The **-r** option should be specified as the digit 1 for master mode when *uucico* is started by a program or *cron. Uux* and *uucp* both queue jobs that will be transferred by *uucico.* It is normally started by the scheduler, *uusched* , but can be started manually; this is done for debugging. For example, the shell *Uutry* starts *uucico* with debugging turned on. A single digit must be used for the **-x** option with higher numbers for more debugging.

The **-i** option defines the interface used with *uucico.* This interface only affects slave mode. Known interfaces are UNIX (default), TLI (basic Transport Layer Interface), and TLIS (Transport Layer Interface with Streams modules, read/write).

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Devconfig
/usr/lib/uucp/Sysfiles
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), Uutry(1M), uucp(1C), uusched(1M), uustat(1C), uux(1C).

NAME

uucleanup – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uucleanup [**-C***time*] [**-W***time*] [**-X***time*] [**-m***string*] [**-o***time*] [**-s***system*]

DESCRIPTION

The *uucleanup* command will scan the spool directories for old files and take appropriate action to remove them in a useful way:

Inform the requestor of send/receive requests for systems that cannot be reached.

Return mail, which cannot be delivered, to the sender.

Delete or execute rnews for rnews type files (depending on where the news originated--locally or remotely).

Remove all other files.

In addition, there is provision to warn users of requests that have been waiting for a given number of days (default 1). Note that *uucleanup* will process as if all option *times* were specified to the default values unless *time* is specifically set.

The following options are available.

-C*time* Any **C.** files greater or equal to *time* days old will be removed with appropriate information to the requestor. (default 7 days)

-D*time* Any **D.** files greater or equal to *time* days old will be removed. An attempt will be made to deliver mail messages and execute rnews when appropriate. (default 7 days)

-W*time* Any **C.** files equal to *time* days old will cause a mail message to be sent to the requestor warning about the delay in contacting the remote. The message includes the *JOBID,* and in the case of mail, the mail message. The administrator may include a message line telling whom to call to check the problem (**-m** option). (default 1 day)

-X*time* Any **X.** files greater or equal to *time* days old will be removed. The **D.** files are probably not present (if they were, the **X.** could get executed). But if there are **D.** files, they will be taken care of by D. processing. (default 2 days)

-m*string* This line will be included in the warning message generated by the **-W** option.

-o*time* Other files whose age is more than *time* days will be deleted. (default 2 days) The default line is "See your local administrator to locate the problem."

-s*system* Execute for *system* spool directory only.

-x*debug_level*
The **-x** debug level is a single digit between 0 and 9; higher numbers give more detailed debugging information. (If **uucleanup** was compiled with -DSMALL, no debugging output will be available.)

This program is typically started by the shell *uudemon.cleanup,* which should be started by *cron*(1M).

FILES

/usr/lib/uucp	directory with commands used by *uucleanup* internally
/usr/spool/uucp	spool directory

SEE ALSO

cron(1M), uucp(1C), uux(1C).

NAME

uucp, uulog, uuname – UNIX system to UNIX system copy

SYNOPSIS

uucp [options] source-files destination-file
uulog [options] **-s**system
uulog [options] system
uulog [options] **-f**system
uuname [**-l**] [**-c**]

DESCRIPTION

uucp

The *uucp* command 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 that *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 to the destination. See WARNINGS and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to forward information (see WARNINGS below for restrictions).

The following shell metacharacters are disallowed in *system-name*:

```
;  &  |    ^  <  >  '  '  "  ?  *  [  ]  (  )  {  }  $  #  \  ~
```

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 ~*/destination* where *destination* is appended to **/usr/spool/uucppublic**. (NOTE: This destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example **~/dan/** as the destination will make the directory **/usr/spool/uucppublic/dan** if it does not exist and put the requested file(s) in that directory.)

(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.

The *uucp* command preserves execute permissions across the transmission and gives 0666 read and write permissions [see *chmod*(2)].

The following options are interpreted by *uucp*:

-c Do not copy local file to the spool directory for transfer to the remote machine (default).

-C Force the copy of local files to the spool directory for transfer.

-d Make all necessary directories for the file copy (default).

-f Do not make intermediate directories for the file copy.

-g*grade* *Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

-m Send mail to the requester when the copy is completed.

-n*user* Notify *user* on the remote system that a file was sent.

-r Do not start the file transfer, just queue the job.

-s*file* Report status of the transfer to *file.* Note that the *file* must be a full path name.

-x*debug_level*
Produce debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information. (Debugging will not be available if **uucp** was compiled with -DSMALL.)

uulog

The *uulog* command queries a log file of *uucp* or *uuxqt* transactions in a file

/usr/spool/uucp/.Log/uucico/*system,*

or

/usr/spool/uucp/.Log/uuxqt/*system.*

The options cause *uulog* to print logging information:

-s*sys* Print information about file transfer work involving system *sys.*

-f*system* Does a "tail **-f**" of the file transfer log for *system.* (You must hit BREAK to exit this function.) Other options used in conjunction with the above:

-x Look in the *uuxqt* log file for the given system.

-*number* Indicates that a "tail" command of *number* lines should be executed.

uuname

The *uuname* command lists the names of systems known to *uucp.* The **-c** option returns the names of systems known to *cu.* (The two lists are the same, unless your machine is using different *Systems* files for *cu* and *uucp.* See the *Sysfiles* file.) The **-l** option returns the local system name.

FILES

/usr/spool/uucp	spool directories
/usr/spool/uucppublic/*	public directory for receiving and sending (**/usr/spool/uucppublic**)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uustat(1C), uux(1C), uuxqt(1M).
chmod(2) in the *Programmer's Reference Manual.*

WARNINGS

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to ~/).

All files received by *uucp* will be owned by *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.

The forwarding of files through other systems may not be compatible with the previous version of *uucp*. If forwarding is used, all systems in the route must have the same version of *uucp*.

BUGS

Protected files and files that are in protected directories that are owned by the requester can be sent by *uucp*. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

NAME

uugetty – set terminal type, modes, speed, and line discipline

SYNOPSIS

/usr/lib/uucp/uugetty [**-h**] [**-t** timeout] [**-r**] line
[speed [type [linedisc]]]
/usr/lib/uucp/uugetty -c file

DESCRIPTION

The *uugetty* command is identical to *getty*(1M) but changes have been made to support using the line for *uucico, cu,* and *ct;* that is, the line can be used in both directions. The *uugetty* will allow users to log in, but if the line is free, *uucico, cu,* or *ct* can use it for dialing out. The implementation depends on the fact that *uucico, cu,* and *ct* create lock files when devices are used. When the "open()" returns (or the first character is read when **-r** option is used), the status of the lock file indicates whether the line is being used by *uucico, cu, ct,* or someone trying to log in. Note that in the **-r** case, several <carriage-return> characters may be required before the log in message is output. The human users will be able to handle this slight inconvenience. *Uucico* trying to login will have to be told by using the following login script:

```
" "   \r\d\r\d\r\d\r in:--in: . . .
```

where the . . . is whatever would normally be used for the login sequence.

An entry for an intelligent modem or direct line that has a *uugetty* on each end must use the **-r** option. (This causes *uugetty* to wait to read a character before it puts out the login message, thus preventing two uugettys from looping.) If there is a *uugetty* on one end of a direct line, there must be a *uugetty* on the other end as well. Here is an **/etc/inittab** entry using *uugetty* on an intelligent modem or direct line:

```
30:2:respawn:/usr/lib/uucp/uugetty –r –t 60 tty12 1200
```

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

ct(1C), cu(1C), getty(1M), init(1M), login(1), uucico(1M), tty(7).
ioctl(2), gettydefs(4), inittab(4) in the *Programmer's Reference Manual.*

BUGS

Ct will not work when uugetty is used with an intelligent modem such as Penril or Ventel.

NAME

uusched – the scheduler for the uucp file transport program

SYNOPSIS

/usr/lib/uucp/uusched [**-x** debug_level] [**-u** debug_level]

DESCRIPTION

The *uusched* command is the *uucp* file transport scheduler. It is usually started by the daemon *uudemon.hour* that is started by *cron*(1M) from an entry in **/usr/spool/cron/crontab**:

```
39 * * * * /bin/su uucp -c "/usr/lib/uucp/uudemon.hour > /dev/null"
```

The two options are for debugging purposes only; **-x** *debug_level* will output debugging messages from *uusched* and **-u** *debug_level* will be passed as **-x** *debug_level* to *uucico*. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), uucico(1M), uucp(1C), uustat(1C), uux(1C).

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [**-a**]
uustat [**-m**]
uustat [**-p**]
uustat [**-q**]
uustat [**-k***jobid*]
uustat [**-r***jobid*]
uustat [**-s***system*] [**-u***user*]

DESCRIPTION

The *uustat* command will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. Only one of the following options can be specified with *uustat* per command execution:

-a Output all jobs in queue.

-m Report the status of accessibility of all machines.

-p Execute a "ps -flp" for all the process-ids that are in the lock files.

-q List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of **C** or **X** files, it is the age in days of the oldest **C./X.** file for that system. The Retry field represents the number of hours until the next possible call. The Count is the number of failure attempts. NOTE: for systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute. As an example of the output produced by the **-q** option:

```
eagle     3C    04/07-11:07   NO DEVICES AVAILABLE
mh3bs3    2C    07/07-10:42   SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

-k*jobid* Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

-r*jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the deamon.

Either or both of the following options can be specified with *uustat*:

-s*sys* Report the status of all *uucp* requests for remote system *sys*.
-u*user* Report the status of all *uucp* requests issued by *user*.

Output for both the **-s** and **-u** options has the following format:

```
eaglen0000  4/07-11:01:03      (POLL)
eagleN1bd7  4/07-11:07         Seagledan522 /usr/dan/A
eagleC1bd8  4/07-11:07         Seagledan59 D.3b2al2ce4924
            4/07-11:07         Seagledanrmail mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an 'S' or 'R' depending on whether the job is to send or request a file. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (*rmail* – the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., D.3b2alce4924) that is created for data files associated with remote executions (*rmail* in this example).
When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user.

FILES

/usr/spool/uucp/* spool directories

SEE ALSO

uucp(1C).

NAME

uuto, uupick – public UNIX system to UNIX system file copy

SYNOPSIS

uuto [options] source-files destination
uupick [**-s** system]

DESCRIPTION

The *uuto* command 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*). *User* 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. By default this directory is /usr/spool/uucppublic. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by *mail*(1) of the arrival of files.

The *uupick* command 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*] **?**

The *uupick* command 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*. If *dir* is not specified as a complete path name (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
a [*dir*]	Same as **m** except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.

*	Print a command summary.

The *uupick* command invoked with the **-s***system* option will only search the PUBDIR for files sent from *system*.

FILES

PUBDIR /usr/spool/uucppublic   public directory

SEE ALSO

mail(1), uucleanup(1M), uucp(1C), uustat(1C), uux(1C).

WARNINGS

In order to send files that begin with a dot (e.g., .profile) the files must by qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

NAME

uux – UNIX system to UNIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

The *uux* command 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: For security reasons, most installations limit the list of commands executable on behalf of an incoming request from *uux*, permiting only the receipt of mail [see *mail*(1)]. (Remote execution permissions are defined in **/usr/lib/uucp/Permissions**.)

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

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/file1   pwba!/a4/dan/file2   >
!~/dan/file.diff"
```

will get the *file1* and *file2* files from the "usg" and "pwba" machines, execute a *diff*(1) command and put the results in *file.diff* in the local PUBDIR/dan/ directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

The *uux* command will attempt to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!cut -f1 b!/usr/file \(c!/usr/file\)
```

gets /usr/file from system "b" and sends it to system "a", performs a *cut* command on that file, and sends the result of the *cut* command to system "c".

The *uux* command will notify you if the requested command on the remote system was disallowed. This notification can be turned off by the **-n** option. 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*.
-a*name*	Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)
-b	Return whatever standard input was provided to the *uux* command if the exit status is non-zero.
-c	Do not copy local file to the spool directory for transfer to the remote machine (default).
-C	Force the copy of local files to the spool directory for transfer.
-g*grade*	*Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.
-j	Output the jobid ASCII string on the standard output, which is the job identification. This job identification can be used by *uustat* to obtain the status or terminate a job.
-n	Do not notify the user if the command fails.
-p	Same as –: The standard input to *uux* is made the standard input to the *command-string*.
-r	Do not start the file transfer, just queue the job.
-s*file*	Report status of the transfer in *file*.
-x*debug_level*	Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.
-z	Send success notification to the user.

FILES

/usr/lib/uucp/spool	spool directories
/usr/lib/uucp/Permissions	remote execution permissions
/usr/lib/uucp/*	other data and programs

SEE ALSO

cut(1), mail(1), uucp(1C), uustat(1C).

WARNINGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the *uucp* system. All files required for the execution will be put into this directory unless they already reside on that machine.

Therefore, the simple file name (without path or machine reference) must be unique within the *uux* request. The following command will NOT work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

but the command

```
uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work (if *diff* is a permitted command).

BUGS

Protected files and files that are in protected directories that are owned by the requester can be sent in commands using *uux*. However, if the requester is root, and the directory is not searchable by "other", the request will fail.

NAME

uuxqt – execute remote command requests

SYNOPSIS

/usr/lib/uucp/uuxqt [**-s** system] [**-x** debug_level]

DESCRIPTION

The *uuxqt* command executes remote job requests from remote systems generated by the use of the *uux* command. (*Mail* uses *uux* for remote mail requests.) *uuxqt* searches the spool directories looking for X. files. For each X. file, *uuxqt* checks to see if all the required data files are available and accessible, and file commands are permitted for the requesting system. The *Permissions* file is used to validate file accessibility and command execution permission.

There are two environment variables that are set before the *uuxqt* command is executed:
UU_MACHINE is the machine that sent the job (the previous one).
UU_USER is the user that sent the job.
These can be used in writing commands that remote systems can execute to provide information, auditing, or restrictions.

The **-x** *debug_level* is a single digit between 0 and 9. Higher numbers give more detailed debugging information.

FILES

/usr/lib/uucp/Permissions
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*

SEE ALSO

mail(1), uucico(1M). uucp(1C), uustat(1C), uux(1C).

NAME

vi, view, vedit – screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [**-t** tag] [**-r** file] [**-L**] [**-wn**] [**-R**] [**-x**] [**-C**] [**-c** command] file ...
view [**-t** tag] [**-r** file] [**-L**] [**-wn**] [**-R**] [**-x**] [**-C**] [**-c** command] file ...
vedit [**-t** tag] [**-r** file] [**-L**] [**-wn**] [**-R**] [**-x**] [**-C**] [**-c** command] file ...

DESCRIPTION

vi (visual) is a display-oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa. The visual commands are described on this manual page; how to set options (like automatically numbering lines and automatically starting a new output line when you type carriage return) and all *ex*(1) line editor commands are described on the *ex*(1) manual page.

When using *vi*, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

Invocation Options

The following invocation options are interpreted by *vi* (previously documented options are discussed in the **NOTES** section at the end of this manual page):

-t *tag*	Edit the file containing the *tag* and position the editor at its definition.
-r *file*	Edit *file* after an editor or system crash. (Recovers the version of *file* that was in the buffer when the crash occurred.)
-L	List the name of all files saved as the result of an editor or system crash.
-w*n*	Set the default window size to *n*. This is useful when using the editor over a slow-speed line.
-R	**Read-only** mode; the **read-only** flag is set, preventing accidental overwriting of the file.
-x	Encryption option; when used, *vi* simulates the **X** command of *ex*(1) and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of *crypt*(1). The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **-x** option. [See *crypt*(1)]. Also, see the **WARNING** section at the end of this manual page.
-C	Encryption option; same as the **-x** option, except that *vi* simulates the **C** command of *ex*(1). The **C** command is like the **X** command of *ex*(1), except that all text read in is assumed to have been encrypted.
-c *command*	Begin editing by executing the specified editor *command* (usually a search or positioning command).

The *file* argument indicates one or more files to be edited.

The *view* invocation is the same as *vi* except that the **read-only** flag is set.

The *vedit* invocation is intended for beginners. It is the same as *vi* except that the **report** flag is set to 1, the **showmode** and **novice** flags are set, and **magic** is turned off. These defaults make it easier to learn how to use *vi*.

vi Modes

Command	Normal and initial mode. Other modes return to command mode upon completion. **ESC** (escape) is used to cancel a partial command.
Input	Entered by setting any of the following options: **a A i I o O c C s S R .** Arbitrary text may then be entered. Input mode is normally terminated with **ESC** character, or, abnormally, with an interrupt.
Last line	Reading input for **: / ?** or **!**; terminate by typing a carriage return; an interrupt cancels termination.

COMMAND SUMMARY

In the descriptions, CR stands for carriage return and **ESC** stands for the escape key.

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text***ESC**	insert *text*
cw*new***ESC**	change word to *new*
eas**ESC**	pluralize word (end of word; append **s**; escape from input state)
x	delete a character
dw	delete a word
dd	delete a line
3dd	delete 3 lines
u	undo previous change
ZZ	exit *vi*, saving changes
:q!CR	quit, discarding changes
/*text***CR**	search for *text*
^U ^D	scroll up or down
:*cmd***CR**	any *ex* or *ed* command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	**z G |**
scroll amount	**^D ^U**
repeat effect	most of the rest

Interrupting, canceling

ESC	end insert or incomplete cmd
DEL	(delete or rubout) interrupts

File manipulation

ZZ	if file modified, write and exit; otherwise, exit
:wCR	write back changes
:w!CR	forced write, if permission originally not valid
:qCR	quit
:q!CR	quit, discard changes
:e *name***CR**	edit file *name*
:e!CR	reedit, discard changes
:e + *name***CR**	edit, starting at end
:e +*n***CR**	edit starting at line *n*
:e #CR	edit alternate file
:e! #CR	edit alternate file, discard changes
:w *name***CR**	write file *name*
:w! *name***CR**	overwrite file *name*
:shCR	run shell, then return
:!*cmd***CR**	run *cmd*, then return
:nCR	edit next file in arglist
:n *args***CR**	specify new arglist
^G	show current file and line
:ta *tag***CR**	position cursor to *tag*

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a carriage return.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
*n***G**	go to the beginning of the specified line (end default), where *n* is a line number
/pat	next line matching *pat*
?pat	previous line matching *pat*
n	repeat last / or ? command
N	reverse last / or ? command
/pat/+n	n-th line after *pat*
?pat?-n	n-th line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching **( )** { or }

Adjusting the screen

^L	clear and redraw window
^R	clear and redraw window if **^L** is → key
zCR	redraw screen with current line at top of window
z-CR	redraw screen with current line at bottom of window
z.CR	redraw screen with current line at center of window
*/pat/***z-CR**	move *pat* line to bottom of window

z*n*.CR	use *n*-line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

\`\`	move cursor to previous context
''	move cursor to first non-white space in line
m*x*	mark current position with the ASCII lower-case letter *x*
\`*x*	move cursor to mark *x*
'*x*	move cursor to first non-white space in line marked by *x*

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
-	previous line, at first non-white
CR	return, same as +
↓ or **j**	next line, same column
↑ or **k**	previous line, same column

Character positioning

^	first non white-space character
0	beginning of line
$	end of line
h or →	forward
l or ←	backward
^**H**	same as ← (backspace)
space	same as → (space bar)
f*x*	find next *x*
F*x*	find previous **x**
t*x*	move to character prior to next *x*
T*x*	move to character following previous *x*
;	repeat last **f F t** or **T**
,	repeat inverse of last **f F t** or **T**
n|	move to column *n*
%	find matching **(** { **)** or }

Words, sentences, paragraphs

w	forward a word
b	back a word
e	end of word
)	to next sentence
}	to next paragraph
(	back a sentence
{	back a paragraph
W	forward a blank-delimited word
B	back a blank-delimited word
E	end of a blank-delimited word

Corrections during insert

^H	erase last character (backspace)
^W	erase last word
erase	your erase character, same as **^H** (backspace)
kill	your kill character, erase this line of input
\	quotes your erase and kill characters
ESC	ends insertion, back to command mode
DEL	interrupt, terminates insert mode
^D	backtab one character; reset left margin of *autoindent*
^^D	caret (^) followed by control-d (^D); backtab to beginning of line; do not reset left margin of *autoindent*
0^D	backtab to beginning of line; reset left margin of *autoindent*
^V	quote non-printable character

Insert and replace

a	append after cursor
A	append at end of line
i	insert before cursor
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text***ESC**	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, e.g., **dd** to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)
X	delete characters before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked; however, if a buffer is named (using the ASCII lower-case letters **a** - **z**), the text in that buffer is put instead.

3yy	yank 3 lines
3yl	yank 3 characters
p	put back text after cursor
P	put back text before cursor
"*x*p	put from buffer *x*
"*x*y	yank to buffer *x*
"*x*d	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d*p	retrieve *d*'th last delete

AUTHOR

vi and *ex* were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/tmp	default directory where temporary work files are placed; it can be changed using the **directory** option (see the *ex(1)* **set** command)
/usr/lib/terminfo/?/*	compiled terminal description data base
/usr/lib/.COREterm/?/*	subset of compiled terminal description data base

NOTES

Two options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard [see *intro*(1)]. A **-r** option that is not followed with an option-argument has been replaced by **-L** and +*command* has been replaced by **-c** *command*.

SEE ALSO

ed(1), edit(1), ex(1).
User's Guide.
Editing Guide.
curses/terminfo chapter of the *Programmer's Guide*.

WARNINGS

The encryption options are provided with the Security Administration Utilities package, which is available only in the United States.

Tampering with entries in */usr/lib/.COREterm/?/** or */usr/lib/terminfo/?/** (for example, changing or removing an entry) can affect programs such as *vi*(1) that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

BUGS

Software tabs using ^**T** work only immediately after the *autoindent.*

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

NAME

volcopy – make literal copy of file system

SYNOPSIS

/etc/volcopy [options] fsname srcdevice volname1 destdevice volname2

DESCRIPTION

The *volcopy* command makes a literal copy of the file system using a block-size matched to the device. *Options* are:

-a invoke a verification sequence requiring a positive operator response instead of the standard 10-second delay before the copy is made

-s (default) invoke the **DEL if wrong** verification sequence.

The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. If the file system is too large to fit on one reel, *volcopy* will prompt for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two or more drives. If *volcopy* is interrupted, it will ask if the user wants to quit or wants a shell. In the latter case, the user can perform other operations (e.g., *labelit*) and return to *volcopy* by exiting the new shell.

The *fsname* argument represents the mounted name (e.g., **root**, **u1**, etc.) of the filsystem being copied.

The *srcdevice* or *destdevice* should be the physical disk section or tape (e.g.: **/dev/dsk/0s1** etc.).

The *volname* is the physical volume name (e.g.: **pk3**, **t0122**, etc.) and should match the external label sticker. Such label names are limited to six or fewer characters. *Volname* may be **-** to use the existing volume name.

Srcdevice and *volname1* are the device and volume from which the copy of the file system is being extracted. *Destdevice* and *volname2* are the target device and volume.

Fsname and *volname* are recorded in the last 12 characters of the super block (**char fsname[6], volname[6];**).

FILES

/etc/log/filesave.log a record of file systems/volumes copied

SEE ALSO

labelit(1M), sh(1).
fs(4) in the *Programmer's Reference Manual.*

NAME

wait – await completion of process

SYNOPSIS

wait [*n*]

DESCRIPTION

Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

The shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

CAVEAT

If you get the error message *cannot fork, too many processes*, try using the *wait*(1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

If *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

wall – write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

The *wall* command reads its standard input until an end-of-file. It then sends this message to all currently logged-in users preceded by:

Broadcast Message from ...

It is used to warn all users, typically prior to shutting down the system.

The sender must be super-user to override any protections the users may have invoked [see *mesg*(1)].

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

"Cannot send to ..." when the open on a user's tty file fails.

NAME

wc – word count

SYNOPSIS

wc [**-lwc**] [names]

DESCRIPTION

The *wc* command 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

who – who is on the system

SYNOPSIS

who [**-uTlHqpdbrtas**] [file]

who am i

who am I

DESCRIPTION

The *who* command 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 system user. It examines the **/etc/utmp** file at login time to obtain its information. If *file* is given, that file [which must be in *utmp*(4) format] is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

The *who* command with the **am i** or **am I** option identifies the invoking user.

The general format for output is:

name [state] line time [idle] [pid] [comment] [exit]

The *name, line,* and *time* information is produced by all options except **-q**; the *state* information is produced only by **-T**; the *idle* and *pid* information is produced only by **-u** and **-l**; and the *comment* and *exit* information is produced only by **-a**. The information produced for **-p**, **-d**, and **-r** is explained during the discussion of each option, below.

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 *idle* column contains 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 **-s** 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 does not exist.

-H This option will print column headings above the regular output.

-q This is a quick *who,* displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

-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 *idle* 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. In addition, it produces the process termination status, process id, and process exit status [see *utmp*(4)] under the *idle, pid,* and *comment* headings, respectively.

-t This option indicates the last change to the system clock [via the *date*(1) command] by **root**. See *su*(1M).

-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.

Note to the super-user: after a shutdown to the single-user state, *who* returns a prompt; the reason is that since **/etc/utmp** is updated at login time and there is no login in single-user state, *who* cannot report accurately on this state. *who am i,* however, returns the correct information.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), init(1M), login(1), mesg(1), su(1M).
wait(2), inittab(4), utmp(4) in the *Programmer's Reference Manual.*

NAME

whodo – who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

The *whodo* command produces formatted and dated output from information in the */etc/utmp* and */etc/ps_data* files.

The display is headed by the date, time, and machine name. For each user logged in, device name, user-id and login time is shown, followed by a list of active processes associated with the user-id. The list includes the device name, process-id, cpu minutes and seconds used, and process name.

EXAMPLE

The command:

```
whodo
```

produces a display like this:

```
Tue Mar 12 15:48:03 1985
bailey

tty09   mcn     8:51
   tty09  28158   0:29 sh

tty52   bdr     15:23
   tty52  21688   0:05 sh
   tty52  22788   0:01 whodo
   tty52  22017   0:03 vi
   tty52  22549   0:01 sh

xt162   lee     10:20
   tty08   6748   0:01 layers
   xt162   6751   0:01 sh
   xt163   6761   0:05 sh
   tty08   6536   0:05 sh
```

FILES

/etc/passwd
/etc/ps_data
/etc/utmp

SEE ALSO

ps(1), who(1).

NAME

write – write to another user

SYNOPSIS

write user [line]

DESCRIPTION

The *write* command 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, an interrupt is sent, or the recipient has executed "mesg n". 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 writable 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, such as *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), pr(1), sh(1), who(1).

DIAGNOSTICS

"user is not logged on" if the person you are trying to *write* to is not logged on.

"Permission denied" if the person you are trying to *write* to denies that permission (with *mesg*).

"Warning: cannot respond, set mesg -y" if your terminal is set to *mesg n* and the recipient cannot respond to you.

"Can no longer write to user" if the recipient has denied permission (*mesg n*) after you had started writing.

NAME

wtinit – object downloader for the 5620 DMD terminal

SYNOPSIS

/usr/lib/layersys/wtinit [**-d**] [**-p**] file

DESCRIPTION

The *wtinit* utility downloads the named *file* for execution in the AT&T TELETYPE 5620 DMD terminal connected to its standard output. *file* must be a DMD object file. *wtinit* performs all necessary bootstrap and protocol procedures.

There are two options.

-d Prints out the sizes of the text, data, and bss portions of the downloaded *file* on standard error.

-p Prints the down-loading protocol statistics and a trace on standard error.

The environment variable **JPATH** is the analog of the shell's **PATH** variable to define a set of directories in which to search for *file*.

If the environment variable **DMDLOAD** has the value **hex**, *wtinit* will use a hexadecimal download protocol that uses only printable characters.

Terminal Feature Packages for specific versions of AT&T windowing terminals will include terminal-specific versions of *wtinit* under those installation sub-directories. */usr/lib/layersys/wtinit* is used for *layers*(1) initialization only when no Terminal Feature Package is in use.

EXIT STATUS

Returns **0** upon successful completion, **1** otherwise.

WARNING

Standard error should be redirected when using the **-d** or **-p** options.

SEE ALSO

layers(1).

NAME

xargs – construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

The *xargs* command 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:

-l*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. The value **-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.

The *xargs* command 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 example 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 example 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
```

SEE ALSO

sh(1).

NAME

xtd – extract and print xt driver link structure

SYNOPSIS

xtd [**-f**] [-n ...]

DESCRIPTION

The *xtd* command is a debugging tool for the *xt*(7) driver. It performs an **XTIOCDATA** *ioctl*(2) call on its standard input file to extract the *Link* data structure for the attached group of channels. This call will fail if data extraction has not been configured in the driver or the standard input is not attached to an *xt*(7) channel. The data are printed one item per line on the standard output. The output should probably be formatted via **pr -3**.

The optional flags affect output as follows:

–n *n* is a number in the range 0 to 7. Channel *n* is included in the list of channels to be printed. The default prints all channels, whereas the occurrence of one or more channel numbers implies a subset.

-f Causes a "formfeed" character to be put out at the end of the output, for the benefit of page-display programs.

EXIT STATUS

Returns **0** upon successful completion; **1** otherwise.

SEE ALSO

xts(1M), xtt(1M), ioctl(2), xtproto(5) in the *Programmer's Reference Manual.*
pr(1), xt(7).

NAME

xts – extract and print xt driver statistics

SYNOPSIS

xts [**-f**]

DESCRIPTION

The *xts* command is a debugging tool for the *xt*(7) driver. It performs an **XTIOCSTATS** *ioctl*(2) call on its standard input file to extract the accumulated statistics for the attached group of channels. This call will fail if statistics have not been configured in the driver, or the standard input is not attached to an *xt*(7) channel. The statistics are printed one item per line on the standard output.

-f Causes a "formfeed" character to be put out at the end of the output, for the benefit of page-display programs.

EXIT STATUS

Returns **0** upon successful completion; **1** otherwise.

SEE ALSO

xt(7).
xtd(1M), xtt(1M), ioctl(2), xtproto(5) in the *Programmer's Reference Manual.*

NAME

xtt – extract and print xt driver packet traces

SYNOPSIS

xtt [-f] [-o]

DESCRIPTION

The *xtt* command is a debugging tool for the *xt*(7) driver. It performs an **XTIOCTRACE** *ioctl*(2) call on its standard input file to turn on tracing and extract the circular packet trace buffer for the attached group of channels. This call will fail if tracing has not been configured in the driver, or the standard input is not attached to an *xt*(7) channel. The packets are printed on the standard output.

The optional flags are:

-f	Causes a "formfeed" character to be put out at the end of the output, for the benefit of page-display programs.
-o	Turns off further driver tracing.

EXIT STATUS

Returns **0** upon successful completion; **1** otherwise.

NOTE

If driver tracing has not been turned on for the terminal session by invoking *layers*(1) with the **-t** option, *xtt* will not generate any output the first time it is executed.

SEE ALSO

xtd(1M), xts(1M), ioctl(2), layers(5) in the *Programmer's Reference Manual*.
layers(1), xt(7).

NAME

intro – introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals and UNIX system device drivers. STREAMS [see *intro*(2)] software drivers, modules, and the STREAMS-generic set of *ioctl*(2) system calls are also described.

For hardware-related files, the names of the entries are generally derived from names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding UNIX system device driver are discussed where applicable.

Disk device file names are in the following format:

/dev/{**r**}**dsk/#s#**

where **r** indicates a raw interface to the disk, the first **#** indicates the drive number, and the second **#** indicates the section number of the partitioned device.

SEE ALSO

Disk/Tape Management in the *Operations/System Administration Guide.*

NAME

asy – asynchronous serial port

DESCRIPTION

The asy driver supports both the system board serial port and an additional serial adapter simultaneously. Up to two serial ports are supported. If an adapter for a port is not installed, an attempt to *open* it will fail. The port can be programmed for speed (50—19200 baud), character length, and parity. Output speed is always the same as input speed. The port behaves as described in *termio*(7).

The asynchronous port is a character-at-a-time device for both input and output. This characteristic both limits the bandwidth which can be achieved over a line, and increases the interrupt loading on the central processor. In particular, file transfer programs such as *uucp*(1) may not function well at speeds over 4800 baud.

The baud rates of the serial adapter programmable baud-rate generator do not correspond exactly with system baud rates. In particular, setting B0 will cause a disconnect, setting EXTA will set 19200 baud, and setting EXTB will set 38400 baud. It is not possible to directly set 2000, 3600, or 7200 baud.

FILES

/dev/tty*

SEE ALSO

signal(2), termio(7).

NAME

clone – open any minor device on a STREAMS driver

DESCRIPTION

clone is a STREAMS software driver that finds and opens an unused minor device on another STREAMS driver. The minor device passed to *clone* during the open is interpreted as the major device number of another STREAMS driver for which an unused minor device is to be obtained. Each such open results in a separate *stream* to a previously unused minor device.

The *clone* driver consists solely of an open function. This open function performs all of the necessary work so that subsequent system calls [including *close(2)*] require no further involvement of *clone.*

clone will generate an ENXIO error, without opening the device, if the minor device number provided does not correspond to a valid major device, or if the driver indicated is not a STREAMS driver.

CAVEATS

Multiple opens of the same minor device cannot be done through the *clone* interface. Executing *stat(2)* on the file system node for a cloned device yields a different result from executing *fstat(2)* using a file descriptor obtained from opening the node.

SEE ALSO

log(7).
STREAMS Programmer's Guide.

NAME

console – console interface

DESCRIPTION

The console provides the operator interface to the computer.

The file */dev/console* is the system console, and refers to an asynchronous serial data line originating from the system board. This special file implements the features described in *termio*(7).

The file */dev/contty* refers to a second asynchronous serial data line originating from the system board. This special file implements the features described in *termio*(7).

FILES

/dev/console
/dev/contty

SEE ALSO

termio(7).

NAME

cram 7 – CMOS RAM interface

DESCRIPTION

The cram driver provides an interface to the 64 bytes of battery backed-up RAM. This memory contains information such as diagnostics and configuration information.

Ioctl Calls

CMOSREAD

This call is used to read the contents of one of the CMOS RAM locations. The argument to the ioctl is the address of a buffer of two unsigned characters, the first of which is the address to be read. The ioctl will fill in the second byte with the data. An address less than 0 or greater than 63 will result in an error, with *errno* set to ENXIO.

CMOSWRITE

This call is used to write a value into one of the CMOS RAM locations. The argument to the ioctl is the address of a buffer of two unsigned characters, the first of which is the address and the second of which is the value to write at that address. An address less than 0 or greater than 63 will result in an error, with *errno* set to ENXIO. Note that only the superuser may open the CMOS RAM device for writing, and that the CMOSWRITE ioctl will fail for any other than the superuser.

FILES

/dev/cram

NAME

disk - random access bulk storage medium

DESCRIPTION

The secondary storage devices used by the system are fixed disks and diskettes. Disks are high-speed rotating magnetic media, which are treated as a collection of concentric rings, known as *tracks.* There are several platters (whose number is represented by *n*) in the fixed disk providing up to two surfaces per platter (or a total of up to 2*n* surfaces); each set of up to 2*n* parallel tracks on these surfaces is considered as a group, known as a *cylinder.* Each track is divided into several *sectors.* A sector is usually the smallest unit which can be transferred to or from the disk. However, the drivers allow *read* or *write* operations of any size to or from any location on the disk, except for raw disks.

Logical Disks

It is often useful to partition fixed physical disks into smaller sections, each of which can hold a separate file system. The disk device driver can therefore divide a physical disk into smaller *logical disks* or *partitions.* Each of these logical sub-disks behaves as if it were a distinct disk. A typical division of a disk into logical sub-disks might be as follows:

/dev/dsk/0s0 represents the entire disk on drive 0
/dev/dsk/0s1 represents the first partition on drive 0
/dev/dsk/0s2 represents the second partition on drive 0
/dev/dsk/1s0 represents the entire disk on drive 1
/dev/dsk/1s1 represents the first partition on drive 1
/dev/dsk/1s2 represents the second partition on drive 1

and similarly for the *raw* (character) logical disks, /dev/rdsk/0s0, /dev/rdsk/0s1, etc.

In fact, more complex arrangements are often created. It is often desirable to have logical disks of different sizes, which are suited to different uses. Similarly, it is often desirable to have several alternative ways of partitioning a single disk. Refer to *install*(8) and *mkpart*(1M) for the details of partitioning of a disk. The philosophy behind partitioning is described in the section on creating file systems in the "Operations Handbook" section of the *System Manager's Guide.*

SEE ALSO

fd(7), hd(7), intro(7), mkpart(1M), fdisk(1M).

NAME

display - system console display

DESCRIPTION

The system console (and user's terminal) is composed of two separate pieces: the keyboard [see *keyboard*(7)] and the display. Because of their complexity, and because there are two possible display interfaces (the monochrome and color/graphics adapters), they are discussed in separate manual entries.

The display normally consists of 25 lines of 80 columns each; 40-column lines are also supported by the color/graphics adapter. Writing characters to the console (**/dev/console**) has an effect which depends on the characters. All characters written to **/dev/console** are first processed by the terminal interface [see *termio*(7)]. For example, mapping new-line characters to carriage return plus new-line, and expanding tabs to spaces, will be done before the following processing:

x	Where *x* is not one of the following, displays *x*.
BEL	Generates a *bell* (audible tone, no modulation).
CR	Places the cursor at column 1 of the current line.
LF, VT	Places the cursor at the same column of the next line (scrolls if the the current line is line 25).
FF	Clears the screen and places the cursor at line 1, column 1.
BS	Depends on the previous character: if a _ (underscore), see below; otherwise, if the cursor is not at column 1, it is moved to the left one position on the same line. If the cursor is at column 1 but not line 1, it is moved to column 79 of the previous line. Finally, if the cursor is at column 1, line 1, it is not moved.
_BS*x*	Sets the *underscore* attribute for the character *x* to be displayed. The *underscore* attribute for the color/graphics adapter is a red background with a white foreground.
ESC*x*	Where *x* is *any* of the 256 possible codes (except for **c** and **[**), displays that value uninterpreted. This is useful for utilizing the full set of graphics available on the display. Note again that the characters are processed through the terminal interface prior to this escape sequence. Therefore, to get some of the possible 256 characters it is necessary that the character not be postprocessed. The easiest way to accomplish this is to turn off OPOST in the *c_oflag* field [see *termio*(7)]; however, this may have other side effects.

The display can be controlled by means of ANSI X3.64 *escape sequences,* which are specific sequences of characters, preceded by the ASCII character ESC. The escape sequences, which work on either the monochrome or color/graphics adapter, are the following:

ESCc	Clears the screen and places the cursor at line 1, column 1.
ESC[*x* @	Insert character—inserts *n* characters at the current cursor position.

ESC[*n* A	Cursor up—moves the cursor up *n* lines (default: *n*=1).
ESC[*n* B	Cursor down—moves the cursor down *n* lines (default: *n*=1).
ESC[*n* C	Cursor right—moves the cursor right *n* columns (default: *n*=1).
ESC[*n* D	Cursor left—moves the cursor left *n* columns (default: *n*=1).
ESC[*n* E	Cursor next line—moves the cursor to column 1 of the next line, then down *n*-1 lines (default: *n*=1).
ESC[*n* F	Cursor previous line—moves the cursor to column 1 of the current line, then up *n* lines (default: *n*=1).
ESC[*n* G	Cursor horizontal position—moves the cursor to column *n* of the current line (default: *n*=1).
ESC[*n* ; *m* H	Position cursor—moves the cursor to column *m* of line *n* (default: *n*=1, *m*=1).
ESC[*n* J	Erase window—erases from the current cursor position to the end of the window if *n*=0, from the beginning of the window to the current cursor position if *n*=1, and the entire window if *n*=2 (default: *n*=0).
ESC[*n* K	Erase line—erases from the current cursor position to the end of the line if *n*=0, from the beginning of the line to the current cursor position if *n*=1, and the entire line if *n*=2 (default: *n*=0).
ESC[*n* L	Inserts *n* lines at the current cursor position (default: *n*=1).
ESC[*n* M	Deletes *n* lines starting at the current cursor position (default: *n*=1).
ESC[*n* P	Deletes *n* characters from a line starting at the current cursor position (default: *n*=1).
ESC[*n* S	Scroll up—scrolls the characters in the current window up *n* lines. The bottom *n* lines are cleared to blanks (default: *n*=1).
ESC[*n* T	Scroll down—scrolls the characters in the current window down *n* lines. The top *n* lines are cleared to blanks (default: *n*=1).
ESC[*n* X	Erase character—erases *n* character positions starting at the current cursor position (default: *n*=1).
ESC[*Ps* ; *Ps*; *m*	Character attributes—each *Ps* is one of the following characters; multiple characters are separated by semicolons. These parameters apply to successive characters being displayed, in an additive manner (e.g., both bold and underscoring can be selected). Only the parameters through 7 apply to the monochrome adapter; all parameters apply to the color/graphics adapter. (Default: *Ps*=0).

Ps	Meaning		
0	all attributes off (normal display) (white foreground with black background)		
1	bold intensity		
4	underscore on (white foreground with red background on color)		
5	blink on		
7	reverse video		
30	black	(gray)	foreground
31	read	(light red)	foreground
32	green	(light green)	foreground
33	brown	(yellow)	foreground
34	blue	(light blue)	foreground
35	magenta	(light magenta)	foreground
36	cyan	(light cyan)	foreground
37	white	(bright white)	foreground
40	black	(gray)	background
41	read	(light red)	background
42	green	(light green)	background
43	brown	(yellow)	background
44	blue	(light blue)	background
45	magenta	(light magenta)	background
46	cyan	(light cyan)	background
47	white	(bright white)	background

Note that for character attributes 30-37, the color selected for foreground will depend on whether the *bold intensity* attribute (1) is currently on. If not, the first color listed will result; otherwise the second color listed will result.

Similarly, for character attributes 40-47, the color selected for background will depend on whether the *blink* attribute (5) is currently on. The color selected for background also depends on whether blinking is enabled in color mode byte or no blinking is selected (see the MODE_BLINK and MODE_BG16 bits in the color mode byte defined below.) If the *blink* attribute is not on, then the first color listed will result. If the *blink* attribute is on, and blinking is enabled, then the first color listed will result and it will blink. If the *blink* attribute is on, and no blinking is enabled, then the second color listed will result.

Ioctl Calls

The following ioctls may be used with either the monochrome or color/graphics adapter.

KDGMODE

This call is used to get the current adapter mode. The argument to the ioctl is the address of one of the following structures, as defined in **<sys/kd.h>**, which will be filled in by the call:

```
struct adtmode {
        unchar  am_capability;    /* type of adapter */
        unchar  am_colmode;       /* color mode register */
        unchar  am_colsel;        /* color select register */
}

/* Values for am_capability */
#define MCAP_UNK            0xff     /* unknown */
#define MCAP_MONO           0x03     /* monochrome adapter */
#define MCAP_COLOR          0x02     /* color adapter, 80x25 */
#define MCAP_COLOR40        0x01     /* color adapter, 40x25 */

/* Values for am_colmode */
#define MODE_TYPE           0x07     /* mask for alpha/graphic
                                     modes */
#define MODE_40             0x00     /* 40x25 alphanumeric */
#define MODE_80             0x01     /* 80x25 alphanumeric */
#define MODE_GRAPH          0x04     /* graphics modes */
#define MODE_GRLRES         0x04     /* 160x100 graphics */
#define MODE_GRMRES         0x05     /* 320x200 graphics */
#define MODE_GRHRES         0x06     /* 640x200 graphics */
#define MODE_BLINK          0x00     /* 8 background colors,
                                     blink */
#define MODE_BG16           0x08     /* 16 background colors,
                                     no blink */
#define MODE_CO             0x00     /* enable color */
#define MODE_BW             0x10     /* disable color */
```

The capability byte is determined when the system is started, depending on the presence of either the monochrome or color/graphics adapter. Only one adapter at a time is supported.

On the monochrome adapter, the am_colmode will always contain the value MODE_80, since this is the only option. For the color/graphics adapter, the other values shown above will be combined to give the state of the color adapter mode register. Note that the bits defined do not match the hardware mode register, but that all combinations described in your hardware techical manual are supported. These modes also affect the adapter's 6845 CRT controller.

The am_colsel contains the value present in the color select register (for the color adapter), or 0 for the monochrome adapter. The bits in this field match the hardware color select register.

KDSCRCTRL

This call is used to turn the display off and on. A non-zero argument to the ioctl will turn the display on; a zero argument will turn it off.

The following ioctl is only valid for the color adapter.

KDSMODE

This call is used to set the adapter mode. The argument to the ioctl is the address of a struct adtmode, as defined above, containing the new values of am_colmode and am_colsel. The value in am_capability is ignored. When the mode type (bits 0, 1, and 2) is changed with this call, the color screen is erased and the cursor returned to the home position. If the mode type is not changed, the screen is not cleared.

Note that if one changes to high-resolution graphics mode and has no border color selected (which is the default), nothing will appear on the display. This is because the *colsel* register selects border color in alphanumeric modes and low-resolution graphics, but selects the foreground color in high-resolution graphics. Thus, all dots will print in black (the color selected) on a black background. Because of this, one should change the border color in the color select register to be something other than black when changing to high-resolution graphics mode.

Color Graphics Support

Although it is possible to write character sequences which set arbitrary bits on the screen in any of the three graphics modes, this mode of operation is not currently supported.

FILES

/dev/console

SEE ALSO

stty(1), ioctl(2), keyboard(7), termio(7).

WARNINGS

It is currently not possible to access the 6845 start address registers. Thus, it is impossible to determine the beginning of the color monitor's screen memory.

The alternate/background color bit (bit 4) of the color select register does not appear to affect background colors in alphanumeric modes.

The low-resolution graphics mode appears to be 80 across by 100 down.

NAME

fd – diskette (floppy disk)

DESCRIPTION

The diskette driver provides access to diskettes as both block and character devices. Diskettes must be formatted before their use [see *format(1)*]. Both 512-byte and 1024-byte sectors with MFM encoding are supported. The driver controls up to two diskette drives with one hard disk drive, or one diskette drive with two hard disk drives. The minor device number specifies both the drive number and the format of the diskette.

Diskette device file names (which correspond to a specific major and minor device) are in the following format:

/dev/{**r**}**dsk/f#**{**dq**}**#**{**d**}{**t**}

where **r** indicates a raw (character) interface to the disk, **f#** is the drive number, **d** or **q** indicates double or quad density (512 or 1024 byte sectors), **#** indicates the number of sectors per track, **d** indicates double-sided, and **t** indicates the entire disk (absence of this letter indicates that the first track of the diskette cannot be accessed).

In order to minimize errors when using diskettes, the driver attempts to assure that the diskette is installed when needed, and that the operations requested have been completed before the device close is completed. In particular, the drive is checked for the presence of a diskette each time a read/write request is made to the drive. If this is not true (either the diskette is not physically present or the door is open), the driver retries the request continually, at five-second intervals. The message:

FD(*n*): diskette not present – please insert

appears after each attempt (the *n* represents the drive number). The INTR and QUIT signals are honored in this case, so that the process accessing the diskette drive in question will receive these signals (unless, of course, the process itself is ignoring them). In particular, if the diskette is removed prematurely, or not inserted soon enough, *no data is lost,* provided the correct diskette is inserted in the drive when the message to do so is displayed.

Ioctl Calls

V_GETPARMS

This call is used to get information about the current drive configuration. The argument to the ioctl is the address of one of the following structures, defined in **<sys/vtoc.h>**, which will be filled in by the ioctl:

```
struct disk_parms {
        char    dp_type;        /* Disk type (see below) */
        unchar  dp_heads;       /* Number of heads */
        ushort  dp_cyls;        /* Number of cylinders */
        unchar  dp_sectors;     /* Number of sectors/track */
        ushort  dp_secsiz;      /* Number of bytes/sector */
                                        /* for this partition: */
        ushort  dp_ptag;        /* Partition tag (not used) */
```

```
                ushort  dp_pflag;       /* Partition flag (not used) */
                ushort  dp_pstartsec;   /* Starting sector number */
                ushort  dp_pnumsec;     /* Number of sectors */
        }

        /* Disk types */
        #define DPT_WINI      1         /* Winchester disk */
        #define DPT_FLOPPY    2         /* Floppy */
        #define DPT_OTHER     3         /* Other type of disk */
        #define DPT_NOTDISK 0           /* Not a disk device */
```

For the floppy driver, the disk type will always be DPT_FLOPPY. The unused fields in the disk_parms structure are only applicable to hard disks; however, returning the same structure from both the hard disk driver and the diskette driver allows programs to be written that can understand either one.

V_FORMAT

This call is used to format tracks on a diskette. The argument passed to the ioctl is the address of one of the following structures, defined in **<sys/vtoc.h>**, containing the starting track, number of tracks, and interleave factor:

```
union io_arg {
        struct {
                ushort  start_trk;      /* first track */
                ushort  num_trks;       /* number of tracks
                                        to format */
                ushort  intlv;          /* interleave factor */
        } ia_fmt;
}
```

Formatting will start at the given track and will continue so that the given number of tracks are formatted, using the given interleave factor.

Note that the file descriptor must refer to the character (raw) special device for the desired drive, and the file must have been opened in exclusive mode (i.e. O_EXCL).

FILES

/dev/dsk/f0d9d, /dev/rdsk/f0d9d, ...
/dev/dsk/f0d9dt, /dev/rdsk/f0d9dt, ...
/dev/dsk/f0q15d, /dev/rdsk/f0q15d, ...
/dev/dsk/f0q15dt, /dev/rdsk/f0q15dt, ...

SEE ALSO

format(1), mkpart(1M), ioctl(2), hd(7).

DIAGNOSTICS

The driver will retry failed transfers up to ten times. If the request still has not succeeded, the driver will display an appropriate message. Errors from the diskette controller, other than the above, are displayed as follows:

FD drv *n*, blk *b*: *drive error message*
FD controller *controller error message*

The first message occurs on an error after a transfer has begun, where *n* is the drive the error occurred on, and *b* is the block number that is being read or written. The *drive error message* is one of the messages appearing in the following list:

"Missing data address mark"
: The diskette may not be formatted properly.

"Cylinder marked bad"
: The accessed cylinder has been marked bad by the formatter.

"Seek error (wrong cylinder)"
: The drive positioned itself at the wrong cylinder when attempting to set up for the requested transfer.

"Uncorrectable data read error"
: A CRC error was detected when attempting to read the requested block from the drive.

"Sector marked bad"
: The accessed sector has been marked bad by the formatter.

"Missing header address mark"
: The diskette may not be formatted properly.

"Write protected"
: A write was attempted to a diskette that is currently write-protected.

"Sector not found"
: The diskette may not be formatted properly.

"Data overrun"
: The system could not keep up with the requested transfer of data. (Should not occur.)

"Header read error"
: The diskette may not be formatted properly.

"Illegal sector specified"
: The driver is confused about the format of the diskette that has been inserted. (Should not occur.)

The second message occurs when there is a controller error during the setup for, or actual transfer of a block. The *controller error message* is one of the messages appearing in the following list:

"command timeout"
: The controller failed to complete the requested command in a reasonable length of time.

"status timeout"
: The controller failed to return its status after a command was completed.

"busy"
: During an attempt to access the controller, a timeout occurred.

NAME

hd – hard (fixed) disk

DESCRIPTION

The hard disk driver supports an IBM disk controller. It can handle up to two hard disk drives with one diskette drive, or one hard disk drive with two diskette drives. The drive characteristics are read from the CMOS RAM at boot time; these characteristics are defined during system setup by using the *setup* program on the AT Diagnostics diskette. The driver determines the layout of the disk dynamically, as described below. It provides block and character (raw) access to the individual partitions of the disk, as well as the entire physical disk.

The minor device number of the device being accessed determines how the drive is treated: the low-order 4 bits determine the partition (0—15), and the fifth bit determines the drive number (0 or 1). Partition 0 represents the entire UNIX partition (as defined by the *fdisk* table). Other partitions are defined by information in the volume table of contents (VTOC). When accessing partition 0, other partition boundaries are ignored, and no bad block mapping occurs. Thus, the user must take care when using the disk in this way.

The full fixed disk is partitioned at two levels: first, sections of the disk to be used by different operating systems are described by the *fdisk* table contained in the first block of the disk. Second, the UNIX system sections of the disk are further partitioned according to information contained in the VTOC, which may be located in any block. [The VTOC is currently in the first block on the second track of the disk; see *mkpart*(1M).] The VTOC also contains information about the non-UNIX system partitions described in the *fdisk* table. When the disk device is opened, the VTOC is read by the driver and is used to fill out its tables of logical disks, assigned by minor device number. The driver does not use the *fdisk* table; however, this table is used by *mkpart*(1M) and by the bootstrap (see below).

Each partition in the *fdisk* table is specified as to its type (e.g., DOS, UNIX system, or other). A partition (file system) is usable by the UNIX system only if its type is correct (e.g., a DOS partition is not usable by the UNIX system, except as a *raw*, non-file system device.)

On each drive, sector 0 contains the first-stage bootstrap and the *fdisk* table. Sector 17 (the first sector on the second track) contains the VTOC, and sector 18 contains the bad block map. The remaining tracks in the first cylinder contain the alternate sectors [although this may be changed through the **/etc/partitions** file; see *mkpart*(1M)]. Thus, the first actual usable partition of the disk starts on the second cylinder.

The *fdisk* table indicates which of the partitions is the 'active', or bootable, partition. When the machine is booted, the first-stage boot code looks in the *fdisk* table for the active partition and jumps to sector 0 of that partition to find the second-stage bootstrap. If the second-stage bootstrap is over one sector in length, it is the responsibility of the second-stage bootstrap to understand this. Note that both the first cylinder (containing the *fdisk* table, first-stage bootstrap, VTOC, and alternate sectors) and the first track of the active partition (containing the second-stage bootstrap) can only be accessed using partition 0, since these tracks are normally not considered part of any other partition in the VTOC.

Bad sectors are mapped out by the driver as follows: The bad block map is read by the driver when the drive is first opened. The map is an array of pairs of numbers, representing a bad sector and its assigned alternate, each entry being an absolute sector number, starting with 0 for the first sector of the disk. There is a software-imposed limit of 62 bad sectors per drive.

Before each I/O operation, the driver looks through the map to determine if any sector in the requested transfer is bad. If there is a bad sector within the request, all the I/O up to the bad sector is done, then the bad sector is remapped, and finally the I/O following the bad sector is done.

Note that this scheme requires running *mkpart*(1M) before bringing up the system from the hard disk for the first time. The *mkpart* program will attempt to optionally write and then read every sector on the disk, looking for sectors where this operation fails. All bad sectors will be placed in alternates map, which is built by *mkpart* and installed on the disk at the same time that the VTOC is installed. If this verification pass is not done, however, the system will still work. Since the driver will notice that the table is empty, it will not attempt to map bad sectors.

In the event that a disk develops bad blocks once the system is running, *mkpart* may run (with the **-A** option) to add the new bad blocks to the map. However, the user may have to restore the file system from the last full dump, depending on where the bad block occurred.

Ioctl Calls

V_CONFIG

This call is used by *mkpart* to reconfigure the drive, so that the drive configuration matches the parameters specified in the **/etc/partitions** file. This is useful because the disk type read from the CMOS RAM is limited to one of 23 types defined in a table in the system BIOS. If the disk installed on the system does not exactly match one of the table entries, the machine is set up using the closest table entry, and *mkpart* will tell the driver the true disk parameters (as defined by the **/etc/partitions** file) by using the V_CONFIG ioctl. The argument to the ioctl is the address of one of the following structures, defined in <**sys/vtoc.h**>, containing the new configuration parameters:

```
union io_arg {
        struct {
                ushort  ncyl;     /* number of cylinders */
                unchar  nhead;    /* heads/cylinder */
                unchar  nsec;     /* sectors/track */
                ushort  secsiz;   /* bytes/sector */
        } ia_cd;
}
```

Note that it is not possible to change the sector size on the hard disk, and that an attempt to do so will result in the ioctl failing, with *errno* set to EINVAL.

V_REMOUNT

This call is used to force the driver to re-read the VTOC on the next open of the drive. It will fail if any partition other than partition 0 is currently open, since changing the partition table information is potentially disastrous for a process using the partition. This is used by *mkpart* when it changes the VTOC, so that the driver will update its internal tables.

V_ADDBAD

This call is used to tell the driver about a bad sector. If the new bad sector is an assigned alternate, the ioctl fails with *errno* set to EINVAL; if it is an unassigned alternate it is removed from the alternates map; if neither of these is true, it is assigned an alternate and added to the map. The argument to the ioctl is the address of one of the following structures, defined in **<sys/vtoc.h>**, with the first two fields filled in; the third field is filled in by the ioctl and returned to the user:

```
union io_arg {
        struct {
                ushort   flags;              /* currently not used */
                daddr_t bad_sector;          /* bad sector number */
                daddr_t new_sector;          /* RETURNED alternate
                                                      assigned */
        } ia_abs;
}
```

V_GETPARMS

This call is used to get information about the current drive configuration. The argument to the ioctl is the address of one of the following structures, defined in **<sys/vtoc.h>**, which will be filled in by the ioctl:

```
struct disk_parms {
        char     dp_type;           /* Disk type (see below) */
        unchar   dp_heads;          /* Number of heads */
        ushort   dp_cyls;           /* Number of cylinders */
        unchar   dp_sectors;        /* Number of sectors/track */
        ushort   dp_secsiz;         /* Number of bytes/sector */
        ushort   dp_ptag;           /* Partition tag */
        ushort   dp_pflag;          /* Partition flag */
        daddr_t dp_pstartsec;       /* Starting absolute sector
                                              number */
        daddr_t dp_pnumsec;         /* Number of sectors */
}

/* Disk types */
#define DPT_WINI      1             /* Winchester disk */
#define DPT_FLOPPY    2             /* Floppy */
#define DPT_OTHER     3             /* Other type of disk */
#define DPT_NOTDISK 0               /* Not a disk device */
```

```
/* Partition tag */
#define  V_BOOT       1        /* bootable partition */
#define  V_ROOT       2        /* root filesystem */
#define  V_SWAP       3        /* swap filesystem */
#define  V_USR        4        /* usr filesystem */
#define  V_BACKUP     5        /* entire disk */
#define  V_ALTS       6        /* alternate sectors */
#define  V_OTHER      7        /* non-UNIX system partition */

/* Partition flag */
#define  V_UNMNT      0x001    /* unmountable partition */
#define  V_RONLY      0x010    /* read only partition */
#define  V_OPEN       0x100    /* partition open */
#define  V_VALID      0x200    /* partition valid to use */
```

For the hard disk driver, the disk type will always be DPT_WINI. Since the structure returned by V_GETPARMS is the same for both the diskette and hard disk drivers, programs may be written to understand either one.

V_FORMAT
This call is used to format tracks on a disk. The argument passed to the ioctl is the address of one of the following structures, defined in **<sys/vtoc.h>**, containing the starting track, number of tracks, and interleave factor:

```
union io_arg {
        struct {
                ushort  start_trk;        /* first track */
                ushort  num_trks;         /* number of tracks
                                                to format */
                ushort  intlv;            /* interleave factor */
        } ia_fmt;
}
```

Note that the file descriptor argument to the ioctl must refer to the character (raw) special device for the desired drive, and the file must have been opened in exclusive mode (i.e., O_EXCL).

Partitions

The *fdisk* table allows partitions to be assigned at cylinder boundaries; however, the VTOC will allow partitions to start on track boundaries. This is used in the bootable UNIX system partition to make the first track (containing the bootstrap code) not be an actual part of the partition. The *fdisk* table allows at most four partitions on a fixed disk, but the VTOC allows the UNIX system portion to be divided into at most 16 partitions. Each partition is identified by a minor device number; the mapping from partition to minor device number is made at the time the disk is first accessed, and is determined by the **/etc/partitions** file. This mapping will remain the same until the **/etc/partitions** file is changed and the *mkpart* program rerun.

Attempts to open file systems for which there are no partitions will fail (non-existent device). Likewise, attempts to mount [see *mount*(8)] partitions that do not contain UNIX file systems will fail.

FILES

/dev/dsk/0s0, ...
/dev/rdsk/0s0, ...

SEE ALSO

fdisk(1M), mkpart(1M), ioctl(2), fs(4), fd(7).

DIAGNOSTICS

The driver will retry failed transfers up to ten times depending on the error type. Certain errors are not retried. The driver will display an appropriate message upon encountering an error during the transfer. Error types that are retried are indicated in the table below. Errors from the fixed disk controller are displayed as follows:

HD error: drive *n*, cyl *c*, head *h*, sector *s*: *drive error message*
HD controller: *controller error message*

The first message occurs on an error after a transfer has begun, where *n* is the drive the error occurred on, *c* is the cylinder, *h* is the head, and *s* is the sector being read or written. The *drive error message* is one of the messages appearing in the following list:

"Track 0 not found"
The disk may not be formatted properly.

"Uncorrectable data read error"
The controller detected a CRC error when attempting to read the requested block.

"Data address mark not found"
The disk may not be formatted properly.

"Sector not found"
The disk may not be formatted properly.

"Command aborted"
The controller did not complete execution of a command.

"Bad track flag detected"
The block requested has been marked bad, but does not appear in the bad block map.

The second message occurs when there is a controller error during the setup for, or actual transfer of a block. The *controller error message* is one of the messages appearing in the following list:

"command aborted"
The controller failed to complete the requested command.

"write fault"
The controller detected some error on the hard disk drive.

"stays busy"
During an attempt to access the controller, a timeout occurred.

There is one additional message which indicates a controller-corrected error occurred:

NOTE: Soft read error corrected by ECC algorithm: unit *n*, sector *s*

where *n* is the drive the error occurred on, and *s* is the sector being read. This warning indicates that the controller's error-correction algorithm successfully recovered from an error. This may be a symptom of a sector going bad. If this message appears several times for the same sector, that sector should probably be marked bad.

WARNINGS

The VTOC and alternate sector mapping scheme requires that no bad sectors occur in cylinder 0. The *mkpart* program will issue a fatal error message when it attempts to configure a drive where there are bad sectors in the first cylinder. Also, since the second-stage bootstrap must be installed on the first track of the bootable partition, this track must also contain no bad sectors.

NAME

keyboard – system console keyboard

DESCRIPTION

The system console (and user's terminal) is composed of two separate pieces: the keyboard and the display [see *display*(7)]. Because of their complexity they are discussed in separate manual entries.

The actual code sequence delivered to the terminal input routine [see *termio*(7)] is defined by a set of internal tables in the driver. These tables can be modified by software (see the discussion of ioctl calls below.) In addition, the driver can be instructed not to do translations, delivering the keyboard up/down scan codes directly.

There are four translation tables: normal keys, shifted keys, alt keys, and shifted alt keys. Each table contains 128 16-bit entries, with an entry being made up of flags in the high-order 8 bits and the character code in the low-order 8 bits. The values that can be set in the flag byte, as defined in **<sys/kd.h>**, are as follows:

```
/* Flag bits */
#define  NUMLCK        0x8000  /* key is affected by num lock */
#define  CAPLCK        0x4000  /* key is affected by caps lock */
#define  CTLKEY        0x2000  /* key is affected by control key */

/* Key types */
#define  NORMKEY       0x0000  /* key is a normal key */
#define  SHIFTKEY      0x0100  /* key is a shift key */
#define  BREAKKEY      0x0200  /* key is a break key */
#define  SS2PFX        0x0300  /* prefix key with <ESC> N */
#define  SS3PFX        0x0400  /* prefix key with <ESC> O */
#define  CSIPFX        0x0500  /* prefix key with <ESC> [ */
#define  NOKEY         0x0f00  /* key sends nothing */
```

The tables are indexed by the keyboard scan code received. The table that is used is determined by the state of the following special keys:

ALT This key essentially chooses an alternate keyboard. If it is not depressed, the normal and shifted tables are used; if it is depressed, the alt and shifted alt tables are used.

SHIFT Depending on the ALT key, this key shifts into either the shifted table or the shifted alt table. The default shifted table is set up such that SHIFT will generate the ASCII uppercase characters.

The character code found in the table may be further modified by the following keys:

CTRL Produces the appropriate ASCII control character if the CTLKEY bit is set in the flag byte. The control character is produced by masking off all but the low-order 5 bits of the character code in the table. If the CTLKEY bit is not set, the normal character (the code in the table) is generated. In the default tables, the CTRL key only modifies keys in the normal and shifted tables; it has no effect in the alt or shifted alt tables.

CAPS LOCK
: This is a toggle; it controls whether keys that have the CAPLCK bit set in their flag byte go to the normal or the shifted table. If the CAPLCK bit is not set, the normal character is generated regardless of the state of the CAPS LOCK. The SHIFT key inverts whatever state is indicated by the CAPS LOCK. Thus, if CAPS LOCK is off, SHIFT produces uppercase characters; if CAPS LOCK is on, SHIFT produces lowercase characters. In the default tables, the only keys affected by CAPS LOCK are the alphabetic keys.

NUM LOCK
: This is a toggle; it controls whether keys that have the NUMLCK bit set in their flag byte go to the normal or the shifted table. If the NUMLCK bit is not set, the normal character is generated regardless of the state of the NUM LOCK. The SHIFT key inverts whatever state is indicated by the NUM LOCK. In the default tables, the only keys affected by NUM LOCK are the keypad keys. Note that CAPS LOCK and NUM LOCK do exactly the same thing; the only difference is the set of keys affected.

SCROLL LOCK
: This key is marked as a BREAKKEY in its flag byte in both the shifted and shifted alt tables. This causes it to send BREAK to the terminal handler.

The remaining values for the key type are discussed below:

SHIFTKEY
: This is used to mark the left and right SHIFT keys, the CTRL key, the ALT key, the CAPS LOCK, and the NUM LOCK in the translation tables. User programs will normally not be concerned with this flag.

SS2PFX, SS3PFX, CSIPFX
: These are used to generate codes for the function keys and for the ALT keys. If one of these flags is specified in the translation table, the driver will prefix the character code in the table with <ESC>N, <ESC>O, or <ESC>[respectively, where <ESC> represents the ASCII escape character (1b hex).

NOKEY
: This is used to mark entries that should not generate any character code. Keystroke combinations that index table entries marked with this flag generate nothing.

The following tables describe the codes generated by the default tables for all the keys. Keycodes are the values delivered at the keyboard interface when the corresponding key is struck (the down scan code). Note that when the key is released, the same code is delivered, but with the high-order bit set. Thus, codes 01—7f are down codes, and 81—ff are up codes. The generated codes are the codes delivered to the terminal driver after translation. All numbers are in hexadecimal.

Shifting Keys		*Function*
Ctrl	1d	CTRL
Left Shift	2a	SHIFT
Right Shift	36	SHIFT
Alt	38	ALT
Caps Lock	3a	CAPS LOCK
Num Lock	45	NUM LOCK

SPECIAL KEYS Keyboard		Generated Codes				
Key	Code	Normal	SHIFT	CTRL	ALT	SHIFT ALT
BACKSPACE	0e	08 bs	08 bs	08 bs	08 bs	08 bs
TAB	0f	09 ht	1d gs	09 ht	09 ht	1d gs
RETURN	1c	0d cr	0d cr	0d cr	0d cr	0d cr
SPACE	39	20 sp	20 sp	00 nul	20 sp	20 sp
ESC	01	1b esc	1b esc	1b esc	1b esc	1b esc

ALPHABETIC KEYS Keyboard		Generated Codes				
Key	Code	Normal	SHIFT	CTRL	ALT	SHIFT ALT
a	1e	61 a	41 A	01 soh	1b4e61	1b4e41
b	30	62 b	42 B	02 stx	1b4e62	1b4e42
c	2e	63 c	43 C	03 etx	1b4e63	1b4e43
d	20	64 d	44 D	04 eot	1b4e64	1b4e44
e	12	65 e	45 E	05 enq	1b4e65	1b4e45
f	21	66 f	46 F	06 ack	1b4e66	1b4e46
g	22	67 g	47 G	07 bel	1b4e67	1b4e47
h	23	68 h	48 H	08 bs	1b4e68	1b4e48
i	17	69 i	49 I	09 ht	1b4e69	1b4e49
j	24	6a j	4a J	0a lf	1b4e6a	1b4e4a
k	25	6b k	4b K	0b vt	1b4e6b	1b4e4b
l	26	6c l	4c L	0c ff	1b4e6c	1b4e4c
m	32	6d m	4d M	0d cr	1b4e6d	1b4e4d
n	31	6e n	4e N	0e so	1b4e6e	1b4e4e
o	18	6f o	4f O	0f si	1b4e6f	1b4e4f
p	19	70 p	50 P	10 dle	1b4e70	1b4e50
q	10	71 q	51 Q	11 dc1	1b4e71	1b4e51
r	13	72 r	52 R	12 dc2	1b4e72	1b4e52
s	1f	73 s	53 S	13 dc3	1b4e73	1b4e53
t	14	74 t	54 T	14 dc4	1b4e74	1b4e54
u	16	75 u	55 U	15 nak	1b4e75	1b4e55
v	2f	76 v	56 V	16 syn	1b4e76	1b4e56
w	11	77 w	57 W	17 etb	1b4e77	1b4e57
x	2d	78 x	58 X	18 can	1b4e78	1b4e58
y	15	79 y	59 Y	19 em	1b4e79	1b4e59
z	2c	7a z	5a Z	1a sub	1b4e7a	1b4e5a

NUMERIC AND PUNCTUATION Keyboard		Generated Codes				
Key	Code	Normal	SHIFT	CTRL	ALT	SHIFT ALT
1	02	31 1	21 !	31 1	1b4e31	1b4e21
2	03	32 2	40 @	00 nul	1b4e32	1b4e40
3	04	33 3	23 #	33 3	1b4e33	1b4e23
4	05	34 4	24 $	34 4	1b4e34	1b4e24
5	06	35 5	25 %	35 5	1b4e35	1b4e25
6	07	36 6	5e ^	1e rs	1b4e36	1b4e5e
7	08	37 7	26 &	37 7	1b4e37	1b4e26
8	09	38 8	2a *	38 8	1b4e38	1b4e2a
9	0a	39 9	28 (	39 9	1b4e39	1b4e28
0	0b	30 0	29)	30 0	1b4e30	1b4e29
-	0c	2d -	5f _	1f us	1b4e2d	1b4e5f
=	0d	3d =	2b +	3d =	1b4e3d	1b4e2b
[	1a	5b [	7b {	1b esc	1b4e5b	1b4e7b
]	1b	5d]	7d }	1d gs	1b4e5d	1b4e7d
;	27	3b ;	3a :	3b ;	1b4e3b	1b4e3a
'	28	27 '	22 "	27 '	1b4e27	1b4e22
‘	29	60 ‘	7e ~	1e rs	1b4e60	1b4e7e
\	2b	5c \	7c ¦	1c fs	1b4e5c	1b4e7c
,	33	2c ,	3c <	2c ,	1b4e2c	1b4e3c
.	34	2e .	3e >	2e .	1b4e2e	1b4e3e
/	35	2f /	3f ?	1f us	1b4e2f	1b4e3f

KEYPAD Keyboard		Generated Codes				
Key	Code	Normal	SHIFT	CTRL	ALT	SHIFT ALT
*	37	2a *	2a *	2a *	1b4e2a	1b4e2a
scroll	46	1b5b4d	00 break	1b5b4d	1b5b4d	00 break
home	47	1b5b48	37 7	1b5b48	1b5b48	1b4e37
up arrow	48	1b5b41	38 8	1b5b41	1b5b41	1b4e38
page up	49	1b5b56	39 9	1b5b56	1b5b56	1b4e39
minus	4a	1b5b53	2d -	1b5b53	1b5b53	1b4e2d
left arrow	4b	1b5b44	34 4	1b5b44	1b5b44	1b4e34
5	4c	1b5b47	35 5	1b5b47	1b5b47	1b4e35
right arrow	4d	1b5b43	36 6	1b5b43	1b5b43	1b4e36
plus	4e	1b5b54	2b +	1b5b54	1b5b54	1b4e2b
end	4f	1b5b59	31 1	1b5b59	1b5b59	1b4e31
down arrow	50	1b5b42	32 2	1b5b42	1b5b42	1b4e32
page down	51	1b5b55	33 3	1b5b55	1b5b55	1b4e33
insert	52	1b5b40	30 0	1b5b40	1b5b40	1b4e30
del	53	7f	2e .	7f	7f	1b4e2e
sys req	54	1b5b4c	1b5b4d	1b5b4c	1b5b4c	1b5b4d

FUNCTION KEYS Keyboard		Generated Codes				
Key	Code	Normal	SHIFT	CTRL	ALT	SHIFT ALT
F1	3b	1b4f50	1b4f70	1b4f50	1b4f50	1b4f70
F2	3c	1b4f51	1b4f71	1b4f51	1b4f51	1b4f71
F3	3d	1b4f52	1b4f72	1b4f52	1b4f52	1b4f72
F4	3e	1b4f53	1b4f73	1b4f53	1b4f53	1b4f73
F5	3f	1b4f54	1b4f74	1b4f54	1b4f54	1b4f74
F6	40	1b4f55	1b4f75	1b4f55	1b4f55	1b4f75
F7	41	1b4f56	1b4f76	1b4f56	1b4f56	1b4f76
F8	42	1b4f57	1b4f77	1b4f57	1b4f57	1b4f77
F9	43	1b4f58	1b4f78	1b4f58	1b4f58	1b4f78
F10	44	1b4f59	1b4f79	1b4f59	1b4f59	1b4f79

Scan codes 55 through 7f are reserved and do not correspond to any keys on the keyboard.

Ioctl Calls

KDGKBMODE

This call is used to get the current keyboard mode. It returns one of the following numbers, as defined in <**sys/kd.h**>:

```
#define K_RAW       0x00    /* send up/down scan codes */
#define K_XLATE     0x01    /* translate to ascii */
```

KDSKBMODE

This call is used to set the keyboard mode. The argument to the ioctl is either K_RAW or K_XLATE. By using raw mode, the program can see the raw up/down scan codes from the keyboard. In translate mode, the translation tables are used to generate the appropriate character code.

KDGKBENT

This call is used to read one of the entries in the translation tables. The argument to the ioctl is the address of one of the following structures, defined in <**sys/kd.h**>, with the first two fields filled in:

```
struct kbentry {
        unchar  kb_table;       /* which table to use */
        unchar  kb_index;       /* which entry in table */
        ushort  kb_value;       /* value to get/set */
}

/* Table selectors */
#define K_NORMTAB           0x00    /* normal table */
#define K_SHIFTTAB          0x01    /* shifted table */
#define K_ALTTAB            0x02    /* alt table */
#define K_ALTSHIFTTAB       0x03    /* shifted alt table */
```

The ioctl will get the indicated entry from the indicated table and return it in the third field.

KDSKBENT

This call is used to set an entry in one of the translation tables. It uses the same structure as the KDGKBENT ioctl, but with the third field filled in with the value that should be placed in the translation table. This can be used to partially or completely remap the keyboard.

FILES

/dev/console

SEE ALSO

ioctl(2), display(7), termio(7).

NAME

log – interface to STREAMS error logging and event tracing

DESCRIPTION

log is a STREAMS software device driver that provides an interface for the STREAMS error logging and event tracing processes [*strerr*(1M), *strace*(1M)]. *log* presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit *log* messages; and a subset of *ioctl*(2) system calls and STREAMS messages for interaction with a user level error logger, a trace logger, or processes that need to submit their own *log* messages.

Kernel Interface

log messages are generated within the kernel by calls to the function *strlog*:

```
strlog(mid, sid, level, flags, fmt, arg1, ...)
short mid, sid;
char level;
ushort flags;
char *fmt;
unsigned arg1;
```

Required definitions are contained in **<sys/strlog.h>** and **<sys/log.h>**. *mid* is the STREAMS module id number for the module or driver submitting the *log* message. *sid* is an internal sub-id number usually used to identify a particular minor device of a driver. *level* is a tracing level that allows for selective screening out of low priority messages from the tracer. *flags* are any combination of SL_ERROR (the message is for the error logger), SL_TRACE (the message is for the tracer), SL_FATAL (advisory notification of a fatal error), and SL_NOTIFY (request that a copy of the message be mailed to the system administrator). *fmt* is a *printf(3S)* style format string, except that %s, %e, %E, %g, and %G conversion specifications are not handled. Up to NLOGARGS (currently 3) numeric or character arguments can be provided.

User Interface

log is opened via the clone interface, **/dev/log**. Each open of **/dev/log** obtains a separate *stream* to *log*. In order to receive *log* messages, a process must first notify *log* whether it is an error logger or trace logger via a STREAMS I_STR *ioctl* call (see below). For the error logger, the I_STR *ioctl* has an ic_cmd field of I_ERRLOG, with no accompanying data. For the trace logger, the *ioctl* has an ic_cmd field of I_TRCLOG, and must be accompanied by a data buffer containing an array of one or more struct trace_ids elements. Each trace_ids structure specifies an *mid, sid,* and *level* from which message will be accepted. *strlog* will accept messages whose *mid* and *sid* exactly match those in the trace_ids structure, and whose level is less than or equal to the level given in the trace_ids structure. A value of –1 in any of the fields of the trace_ids structure indicates that any value is accepted for that field.

At most one trace logger and one error logger can be active at a time. Once the logger process has identified itself via the *ioctl* call, *log* will begin sending up messages subject to the restrictions noted above. These messages are obtained via the *getmsg(2)* system call. The control part of this message contains a log_ctl structure, which specifies the *mid, sid, level, flags,* time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, and a sequence number. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of *log* messages can be determined.

Different sequence numbers are maintained for the error and trace logging *streams,* and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic, some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by NLOGARGS words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to *log,* even if it is not an error or trace logger. The only fields of the log_ctl structure in the control part of the message that are accepted are the level and flags fields; all other fields are filled in by *log* before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to NLOGARGS) must be packed one word each, on the next word boundary following the end of the format string.

Attempting to issue an I_TRCLOG or I_ERRLOG when a logging process of the given type already exists will result in the error ENXIO being returned. Similarly, ENXIO is returned for I_TRCLOG *ioctls* without any trace_ids structures, or for any unrecognized I_STR *ioctl* calls. Incorrectly formatted *log* messages sent to the driver by a user process are silently ignored (no error results).

EXAMPLES

Example of I_ERRLOG notification.

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;              /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;

ioctl(log, I_STR, &ioc);
```

Example of I_TRCLOG notification.

```
struct trace_ids tid[2];

tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;
```

```
        tid[1].ti_mid = 1002;
        tid[1].ti_sid = -1;         /* any sub-id will be allowed */
        tid[1].ti_level = -1;       /* any level will be allowed */

        ioc.ic_cmd = I_TRCLOG;
        ioc.ic_timout = 0;
        ioc.ic_len = 2 * sizeof(struct trace_ids);
        ioc.ic_dp = (char *)tid;

        ioctl(log, I_STR, &ioc);
```

Example of submitting a *log* message (no arguments).

```
        struct strbuf ctl, dat;
        struct log_ctl lc;
        char *message = "Don't forget to pick up some milk on the way home'

        ctl.len = ctl.maxlen = sizeof(lc);
        ctl.buf = (char *)&lc;

        dat.len = dat.maxlen = strlen(message);
        dat.buf = message;

        lc.level = 0;
        lc.flags = SL_ERROR|SL_NOTIFY;

        putmsg(log, &ctl, &dat, 0);
```

FILES

/dev/log, <sys/log.h>, <sys/strlog.h>

SEE ALSO

strace(1M), strerr(1M), clone(7).
intro(2), getmsg(2), putmsg(2) in the *Programmer's Reference Manual.*
STREAMS Programmer's Guide.

NAME

lp – parallel printer interface

DESCRIPTION

The lp driver supports both the primary (monochrome) and secondary parallel printer adapters simultaneously. Up to two printers are supported. If an adapter for a printer is not installed, an attempt to *open* it will fail. The *close* waits until all output is completed before returning to the user. The lp driver allows only one process at a time to write to the adapter. If it is already busy, an *open* for writing will return an error. However, the driver allows multiple *open*s to occur if they are *read-only.*

The parallel printer adapters are character devices. The minor device number corresponds to the primary or secondary parallel printer adapter. Thus, minor device 0 corresponds to the primary parallel printer adapter, while minor device 1 corresponds to the secondary adapter.

FILES

/dev/lp*

NAME

mem, kmem – core memory

DESCRIPTION

The file */dev/mem* is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system.

Byte addresses in */dev/mem* are interpreted as memory addresses. References to nonexistent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file */dev/kmem* is the same as */dev/mem* except that kernel virtual memory rather than physical memory is accessed.

The per-process data for the current process begins at 0x80880000.

I/O is not memory mapped, and the per-process data begins at virtual address 0xE0000000.

FILES

/dev/mem
/dev/kmem

WARNING

Some of */dev/kmem* cannot be read because of write-only addresses or unequipped memory addresses.

NAME

null – the null file

DESCRIPTION

Data written on the null special file, */dev/null,* is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

prf – operating system profiler

DESCRIPTION

The special file */dev/prf* provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file */dev/prf* is a pseudo-device with no associated hardware.

FILES

/dev/prf

SEE ALSO

profiler(1M).

NAME

qt – QIC cartridge magnetic tape streamer interface

SYNOPSIS

qt

DESCRIPTION

The format for tape files is described below:

/dev/rmt/c0s0n	no rewind on close, no retension on open
/dev/rmt/c0s0	rewind on close, no retension on open
/dev/rmt/c0s0nr	no rewind on close, retension on open
/dev/rmt/c0s0r	rewind on close, retension on open

These files refer to the Wangtek PC-36 Controller and the QIC-24/QIC-02 basic cartridge tape streamer. Only raw character interface files are provided. When opened for reading or writing, the tape is assumed to be positioned as desired. If the file was retension-on-open, the tape is retensioned before any i/o is performed. When a T_RWD, T_RETENSION, T_LOAD, or T_UNLOAD ioctl is requested after a write, a double end-of-file (double tape mark) is written before the ioctl is executed. When a rewind-on-close file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing and data was written. When a rewind-on-close file is closed, the tape is rewound. If the file is no-rewind-on-close and was opened for writing and data was written, only one EOF is written, and the tape is positioned after the EOF just written. If the file was no-rewind and the file was opened for read-only, the tape is positioned after the EOF following the data just read. The EOF is returned as a zero-length read. By judiciously choosing *qt* files, it is possible to read and write multifile tapes.

A standard tape consists of several 512-byte records terminated by an EOF. To the extent possible, the system treats the tape like any other file. As in other raw devices, seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

Only one process is permitted to have any of the tape files open at a given time, to the extent it is enforceable. Writing after reading is permitted, but reading after writing without an intervening rewind is not. If O_NDELAY is clear, opening a retension-on-open file will block until the retension is complete. If O_NDELAY is set, open will return without delay. Opening a file with O_APPEND set is an error (EINVAL).

The following ioctl's are supported:

T_RETENSION	retension the tape
T_RWD	rewind the tape to BOT
T_LOAD	rewind the tape to BOT
T_UNLOAD	rewind the tape to BOT
T_ERASE	erase the tape and leave it at BOT
T_WRFILEM	write an EOF (tape mark)
T_RST	reset the tape device
T_SFF	skip forward arg files

T_SBF	skip forward arg blocks
T_RDSTAT	read the device status registers into the buffer pointed to by arg.

FILES

/dev/rmt/c0s0n
/dev/rmt/c0s0
/dev/rmt/c0s0nr
/dev/rmt/c0s0r

SEE ALSO

intro(7).

NAME

rtc – real time clock interface

DESCRIPTION

The rtc driver supports the real time clock chip, allowing it to be set with the correct local time, and allowing the time to be read from the chip.

Ioctl Calls

RTCRTIME

This call is used to read the local time from the real time clock chip. The argument to the *ioctl* is the address of a buffer of unsigned characters is defined is **<sys/rtc.h>**). The ioctl will fill in the buffer with the contents of the chip registers. Currently, is 14, and the meanings of the byte registers are as follows:

Register	Contents
0	Seconds
1	Second alarm
2	Minutes
3	Minute alarm
4	Hours
5	Hour alarm
6	Day of week
7	Date of month
8	Month
9	Year
A	Status register A
B	Status register B
C	Status register C
D	Status register D

For further information on the functions of these registers, see your hardware technical reference manual.

RTCSTIME

This call is used to set the time into the real time clock chip. The argument to the *ioctl* is the address of a buffer of unsigned characters is defined is **<sys/rtc.h>**.) These bytes should be the desired chip register contents. Currently, is 10, representing registers 0—9 as shown above. Note that only the superuser may open the real time clock device for writing, and that the *ioctl* will fail for any other than the superuser.

FILES

/dev/rtc

NAME

streamio – STREAMS ioctl commands

SYNOPSIS

#include <stropts.h>
int ioctl (fildes, command, arg)
int fildes, command;

DESCRIPTION

STREAMS [see *intro*(2)] ioctl commands are a subset of *ioctl*(2) system calls which perform a variety of control functions on *streams*. The arguments *command* and *arg* are passed to the file designated by *fildes* and are interpreted by the *stream head*. Certain combinations of these arguments may be passed to a module or driver in the *stream*.

fildes is an open file descriptor that refers to a *stream*. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl*, they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the *stream* referenced by *fildes* is linked below a multiplexer, or if *command* is not a valid value for a *stream*.

Also, as described in *ioctl*, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. This causes subsequent system calls to fail with *errno* set to this value.

COMMAND FUNCTIONS

The following *ioctl* commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH
: Pushes the module whose name is pointed to by *arg* onto the top of the current *stream*, just below the *stream head*. It then calls the open routine of the newly-pushed module. On failure, *errno* is set to one of the following values:

[EINVAL]	Invalid module name.
[EFAULT]	*arg* points outside the allocated address space.
[ENXIO]	Open routine of new module failed.
[ENXIO]	Hangup received on *fildes*.

I_POP
: Removes the module just below the *stream head* of the *stream* pointed to by *fildes*. *arg* should be 0 in an I_POP request. On failure, *errno* is set to one of the following values:

[EINVAL]	No module present in the *stream*.
[ENXIO]	Hangup received on *fildes*.

I_LOOK	Retrieves the name of the module just below the *stream head* of the *stream* pointed to by *fildes*, and places it in a null terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long. An `[#include <sys/conf.h>]` declaration is required. On failure, *errno* is set to one of the following values:

[EFAULT]	*arg* points outside the allocated address space.
[EINVAL]	No module present in *stream*.

I_FLUSH — This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

FLUSHR	Flush read queues.
FLUSHW	Flush write queues.
FLUSHRW	Flush read and write queues.

On failure, *errno* is set to one of the following values:

[ENOSR]	Unable to allocate buffers for flush message due to insufficient STREAMS memory resources.
[EINVAL]	Invalid *arg* value.
[ENXIO]	Hangup received on *fildes*.

I_SETSIG — Informs the *stream head* that the user wishes the kernel to issue the SIGPOLL signal [see *signal*(2) and *sigset*(2)] when a particular event has occurred on the *stream* associated with *fildes*. I_SETSIG supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

S_INPUT	A non-priority message has arrived on a *stream head* read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.
S_HIPRI	A priority message is present on the *stream head* read queue. This is set even if the message is of zero length.
S_OUTPUT	The write queue just below the *stream head* is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.
S_MSG	A STREAMS signal message that contains the SIGPOLL signal has reached the front of the *stream head* read queue.

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value S_HIPRI.

Processes that wish to receive SIGPOLL signals must explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same Stream, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further SIGPOLL signals. On failure, *errno* is set to one of the following values:

[EINVAL]	*arg* value is invalid or *arg* is zero and process is not registered to receive the SIGPOLL signal.
[EAGAIN]	Allocation of a data structure to store the signal request failed.

I_GETSIG
Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of I_SETSIG above. On failure, *errno* is set to one of the following values:

[EINVAL]	Process not registered to receive the SIGPOLL signal.
[EFAULT]	*arg* points outside the allocated address space.

I_FIND
Compares the names of all modules currently present in the *stream* to the name pointed to by *arg*, and returns 1 if the named module is present in the *stream*. It returns 0 if the named module is not present. On failure, *errno* is set to one of the following values:

[EFAULT]	*arg* points outside the allocated address space.
[EINVAL]	*arg* does not contain a valid module name.

I_PEEK
Allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *arg* points to a *strpeek* structure which contains the following members:

```
struct strbuf  ctlbuf;
struct strbuf  databuf;
long           flags;
```

The *maxlen* field in the *ctlbuf* and *databuf strbuf* structures [see *getmsg*(2)] must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to RS_HIPRI, I_PEEK will only look for a priority message on the *stream head* read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the RS_HIPRI flag was set in *flags* and a priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or RS_HIPRI. On failure, *errno* is set to the following value:

[EFAULT]	*arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.
[EBADMSG]	Queued message to be read is not valid for I_PEEK

I_SRDOPT Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

RNORM	Byte-stream mode, the default.
RMSGD	Message-discard mode.
RMSGN	Message-nondiscard mode.

Read modes are described in *read*(2). On failure, *errno* is set to the following value:

[EINVAL] *arg* is not one of the above legal values.

I_GRDOPT Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in *read*(2). On failure, *errno* is set to the following value:

[EFAULT] *arg* points outside the allocated address space.

I_NREAD Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the *ioctl* return value is greater than zero, this indicates that a zero-length message is next on the queue. On failure, *errno* is set to the following value:

[EFAULT] *arg* points outside the allocated address space.

I_FDINSERT Creates a message from user specified buffer(s), adds information about another *stream* and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

arg points to a *strfdinsert* structure which contains the following members:

```
struct strbuf  ctlbuf;
struct strbuf  databuf;
long           flags;
int            fildes;
int            offset;
```

The *len* field in the *ctlbuf strbuf* structure [see *putmsg*(2)] must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fildes* in the *strfdinsert* structure specifies the file descriptor of the other *stream*. *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where I_FDINSERT will store a pointer. This pointer will be the address of the read queue structure of the driver for the *stream* corresponding to *fildes* in the *strfdinsert* structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

flags specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to RS_HIPRI. For non-priority messages, I_FDINSERT will block if the *stream* write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NDELAY is set. Instead, it fails and sets *errno* to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether O_NDELAY has been specified. No partial message is sent. On failure, *errno* is set to one of the following values:

[EAGAIN]	A non-priority message was specified, the O_NDELAY flag is set, and the *stream* write queue is full due to internal flow control conditions.
[ENOSR]	Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.
[EFAULT]	*arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.
[EINVAL]	One of the following: *fildes* in the *strfdinsert* structure is not a valid, open *stream* file descriptor; the size of a pointer plus *offset* is greater than the *len* field for the buffer specified through *ctlptr*; *offset* does not specify a properly aligned location in the data buffer; an undefined value is stored in *flags*.
[ENXIO]	Hangup received on *fildes* of the *ioctl* call or *fildes* in the *strfdinsert* structure.
[ERANGE]	The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of

the topmost *stream* module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

I_FDINSERT can also fail if an error message was received by the *stream head* of the *stream* corresponding to *fildes* in the *strfdinsert* structure. In this case, *errno* will be set to the value in the message.

I_STR Constructs an internal STREAMS ioctl message from the data pointed to by *arg* and sends that message downstream.

This mechanism is provided to send user *ioctl* requests to downstream modules and drivers. It allows information to be sent with the *ioctl* and will return to the user any information sent upstream by the downstream recipient. I_STR blocks until the system responds with either a positive or negative acknowledgment message or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one I_STR can be active on a *stream*. Further I_STR calls will block until the active I_STR completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The O_NDELAY [see *open*(2)] flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int     ic_cmd;       /* downstream command */
int     ic_timout;    /* ACK/NAK timeout */
int     ic_len;       /* length of data arg */
char    *ic_dp;       /* ptr to data arg */
```

ic_cmd is the internal ioctl command intended for a downstream module or driver; and *ic_timout* is the number of seconds (-1 = infinite, 0 = use default, >0 = as specified) an I_STR request will wait for acknowledgment before timing out. *ic_len* is the number of bytes in the data argument and *ic_dp* is a pointer to the data argument. The *ic_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the *stream* can return).

The *stream head* will convert the information pointed to by the *strioctl* structure to an internal ioctl command message and send it downstream. On failure, *errno* is set to one of the following values:

[ENOSR]	Unable to allocate buffers for the *ioctl* message due to insufficient STREAMS memory resources.
[EFAULT]	*arg* points, or the buffer area specified by *ic_dp* and *ic_len* (separately for data sent and data returned) is, outside the allocated address space.
[EINVAL]	*ic_len* is less than 0 or *ic_len* is larger than the maximum configured size of the data part of a message or *ic_timout* is less than -1.
[ENXIO]	Hangup received on *fildes*.
[ETIME]	A downstream *ioctl* timed out before acknowledgment was received.

An I_STR can also fail while waiting for an acknowledgment if a message indicating an error or a hangup is received at the *stream head*. In addition, an error code can be returned in the positive or negative acknowledgment message, in the event the ioctl command sent downstream fails. For these cases, I_STR will fail with *errno* set to the value in the message.

I_SENDFD Requests the *stream* associated with *fildes* to send a message, containing a file pointer, to the *stream head* at the other end of a *stream* pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue [see *intro*(2)] of the *stream head* at the other end of the *stream* pipe to which it is connected. On failure, *errno* is set to one of the following values:

[EAGAIN]	The sending *stream* is unable to allocate a message block to contain the file pointer.
[EAGAIN]	The read queue of the receiving *stream head* is full and cannot accept the message sent by I_SENDFD.
[EBADF]	*arg* is not a valid, open file descriptor.
[EINVAL]	*fildes* is not connected to a *stream* pipe.
[ENXIO]	Hangup received on *fildes*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an I_SENDFD *ioctl* over a *stream* pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

fd is an integer file descriptor. *uid* and *gid* are the user id and group id, respectively, of the sending *stream*.

If O_NDELAY is not set [see *open*(2)], I_RECVFD will block until a message is present at the *stream head*. If O_NDELAY is set, I_RECVFD will fail with *errno* set to EAGAIN if no message is present at the *stream head*.

If the message at the *stream head* is a message sent by an I_SENDFD, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*. On failure, *errno* is set to one of the following values:

[EAGAIN]	A message was not present at the *stream head* read queue, and the O_NDELAY flag is set.
[EBADMSG]	The message at the *stream head* read queue was not a message containing a passed file descriptor.
[EFAULT]	*arg* points outside the allocated address space.
[EMFILE]	NOFILES file descriptors are currently open.
[ENXIO]	Hangup received on *fildes*.

The following two commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK Connects two *streams*, where *fildes* is the file descriptor of the *stream* connected to the multiplexing driver, and *arg* is the file descriptor of the *stream* connected to another driver. The *stream* designated by *arg* gets connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgment message to the *stream head* regarding the linking operation. This call returns a multiplexer ID number (an identifier used to disconnect the multiplexer, see I_UNLINK) on success, and a -1 on failure. On failure, *errno* is set to one of the following values:

[ENXIO]	Hangup received on *fildes*.
[ETIME]	Time out before acknowledgment message was received at *stream head*.
[EAGAIN]	Temporarily unable to allocate storage to perform the I_LINK.
[ENOSR]	Unable to allocate storage to perform the I_LINK due to insufficient STREAMS memory resources.

[EBADF]	*arg* is not a valid, open file descriptor.
[EINVAL]	*fildes stream* does not support multiplexing.
[EINVAL]	*arg* is not a *stream,* or is already linked under a multiplexer.
[EINVAL]	The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given *stream head* is linked into a multiplexing configuration in more than one place.

An I_LINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes.* In addition, an error code can be returned in the positive or negative acknowledgment message. For these cases, I_LINK will fail with *errno* set to the value in the message.

I_UNLINK Disconnects the two *streams* specified by *fildes* and *arg. fildes* is the file descriptor of the *stream* connected to the multiplexing driver. *fildes* must correspond to the *stream* on which the *ioctl* I_LINK command was issued to link the *stream* below the multiplexing driver. *arg* is the multiplexer ID number that was returned by the I_LINK. If *arg* is -1, then all Streams which were linked to *fildes* are disconnected. As in I_LINK, this command requires the multiplexing driver to acknowledge the unlink. On failure, *errno* is set to one of the following values:

[ENXIO]	Hangup received on *fildes.*
[ETIME]	Time out before acknowledgment message was received at *stream head.*
[ENOSR]	Unable to allocate storage to perform the I_UNLINK due to insufficient STREAMS memory resources.
[EINVAL]	*arg* is an invalid multiplexer ID number or *fildes* is not the *stream* on which the I_LINK that returned *arg* was performed.

An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes.* In addition, an error code can be returned in the positive or negative acknowledgment message. For these cases, I_UNLINK will fail with *errno* set to the value in the message.

SEE ALSO

close(2), fcntl(2), intro(2), ioctl(2), open(2), read(2), getmsg(2), poll(2), putmsg(2), signal(2), sigset(2), write(2) in the *Programmer's Reference Manual.*
STREAMS Programmer's Guide.
STREAMS Primer.

DIAGNOSTICS

Unless specified otherwise above, the return value from *ioctl* is 0 upon success and -1 upon failure with *errno* set as indicated.

NAME

sxt – pseudo-device driver

DESCRIPTION

The special file */dev/sxt* is a pseudo-device driver that interposes a discipline between the standard *tty* line disciplines and a real device driver. The standard disciplines manipulate *virtual tty* structures (channels) declared by the */dev/sxt* driver. */Dev/sxt* acts as a discipline manipulating a *real tty* structure declared by a real device driver. The */dev/sxt* driver is currently only used by the *shl*(1) command.

Virtual ttys are named by inodes in the subdirectory **/dev/sxt** and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form **/dev/sxt/??0** (channel 0) and then execute a SXTIOCLINK *ioctl* call to initiate the multiplexing.

Only one channel, the *controlling* channel, can receive input from the keyboard at a time; others attempting to read will be blocked.

There are two groups of *ioctl*(2) commands supported by *sxt*. The first group contains the standard *ioctl* commands described in *termio*(7), with the addition of the following:

TIOCEXCL	Set *exclusive use* mode: no further opens are permitted until the file has been closed.
TIOCNXCL	Reset *exclusive use* mode: further opens are once again permitted.

The second group are commands to *sxt* itself. Some of these may only be executed on channel 0.

SXTIOCLINK — Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

EINVAL	The argument is out of range.
ENOTTY	The command was not issued from a real tty.
ENXIO	*linesw* is not configured with *sxt*.
EBUSY	An SXTIOCLINK command has already been issued for this real *tty*.
ENOMEM	There is no system memory available for allocating the virtual tty structures.
EBADF	Channel 0 was not opened before this call.

SXTIOCSWTCH — Set the controlling channel. Possible errors include:

EINVAL	An invalid channel number was given.
EPERM	The command was not executed from channel 0.

SXTIOCWF	Cause a channel to wait until it is the controlling channel. This command will return the error, *EINVAL,* if an invalid channel number is given.
SXTIOCUBLK	Turn off the **loblk** control flag in the virtual tty of the indicated channel. The error *EINVAL* will be returned if an invalid number or channel 0 is given.
SXTIOCSTAT	Get the status (blocked on input or output) of each channel and store in the *sxtblock* structure referenced by the argument. The error *EFAULT* will be returned if the structure cannot be written.
SXTIOCTRACE	Enable tracing. Tracing information is written to the console. This command has no effect if tracing is not configured.
SXTIOCNOTRACE	Disable tracing. This command has no effect if tracing is not configured.

FILES

/dev/sxt/??[0-7]	Virtual tty devices

SEE ALSO

shl(1), stty(1), termio(7).
ioctl(2), open(2) in the *Programmer's Reference Manual.*

NAME

termio – general terminal interface

DESCRIPTION

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open terminal files; they are opened by *getty* and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, the buffer is flushed and all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character **#** erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character **@** kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR	(Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).
QUIT	(Control-| or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.
SWTCH	(Control-z or ASCII SUB) is used by the job control facility, *shl,* to change the current layer to the control layer.
ERASE	(#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.
KILL	(@) deletes the entire line, as delimited by a NL, EOF, or EOL character.
EOF	(Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.
NL	(ASCII LF) is the normal line delimiter. It cannot be changed or escaped.
EOL	(ASCII NUL) is an additional line delimiter, like NL. It is not normally used.
EOL2	is another additional line delimiter.
STOP	(Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.
START	(Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a *hang-up* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hang-up signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several *ioctl*(2) system calls apply to terminal files. The primary calls use the following structure, defined in <**termio.h**>:

```
#define   NCC        8
struct    termio {
          unsigned   short   c_iflag;     /* input modes */
          unsigned   short   c_oflag;     /* output modes */
          unsigned   short   c_cflag;     /* control modes */
          unsigned   short   c_lflag;     /* local modes */
          char               c_line;      /* line discipline */
          unsigned   char    c_cc[NCC];   /* control chars */
};
```

The special control characters are defined by the array *c_cc*. The relative positions and initial values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	reserved	
7	SWTCH	

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map uppercase to lowercase on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character-framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character will restart output which has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all-bits-clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:

TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags; otherwise, characters are transmitted without change.

If OLCUC is set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2, four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all-bits-clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
B19200	0000016	19200 baud
EXTA	0000016	External A
B38400	0000017	38400 baud
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.
RCV1EN	0010000	
XMT1EN	0020000	
LOBLK	0040000	Block layer output.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set; otherwise, even parity is used.

If CREAD is set, the receiver is enabled; otherwise, no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise, modem control is assumed.

If LOBLK is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise the output generated by that layer will be multiplexed onto the current layer.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters, respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an uppercase letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for:	*use*:
\`	\\'
|	\\!
~	\\^
{	\\(
}	\\)
\\	\\\\

For example, **A** is input as \a, \n as \\n, and \N as \\\n.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit, switch, and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary *ioctl*(2) system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA	Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.
TCSETA	Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.
TCSETAW	Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.
TCSETAF	Wait for the output to drain, then flush the input queue and set the new parameters.

Additional *ioctl*(2) calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK	Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).
TCXONC	Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.
TCFLSH	If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FILES

/dev/tty*

SEE ALSO

stty(1).
fork(2), ioctl(2), setpgrp(2), signal(2) in the *Programmer's Reference Manual.*

NAME

timod – Transport Interface cooperating STREAMS module

DESCRIPTION

timod is a STREAMS module for use with the Transport Interface (TI) functions of the Network Services library. The *timod* module converts a set of *ioctl*(2) calls into STREAMS messages that may be consumed by a transport protocol provider which supports the Transport Interface. This allows a user to initiate certain TI functions as atomic operations.

The *timod* module must be pushed (see *Streams Primer*) onto only a *stream* terminated by a transport protocol provider which supports the TI.

All STREAMS messages, with the exception of the message types generated from the *ioctl* commands described below, will be transparently passed to the neighboring STREAMS module or driver. The messages generated from the following *ioctl* commands are recognized and processed by the *timod* module. The format of the *ioctl* call is:

```
#include <sys/stropts.h>
                -
                -
struct strioctl strioctl;
                -
                -
strioctl.ic_cmd = cmd;
strioctl.ic_timeout = INFTIM;
strioctl.ic_len = size;
strioctl.ic_dp = (char *)buf

ioctl(fildes, I_STR, &strioctl);
```

where, on issuance, *size* is the size of the appropriate TI message to be sent to the transport provider and on return, *size* is the size of the appropriate TI message from the transport provider in response to the issued TI message. *buf* is a pointer to a buffer large enough to hold the contents of the appropriate TI messages. The TI message types are defined in *<sys/tihdr.h>*. The possible values for the *cmd* field are:

TI_BIND	Bind an address to the underlying transport protocol provider. The message issued to the TI_BIND *ioctl* is equivalent to the TI message type T_BIND_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_BIND_ACK.
TI_UNBIND	Unbind an address from the underlying transport protocol provider. The message issued to the TI_UNBIND *ioctl* is equivalent to the TI message type T_UNBIND_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_OK_ACK.
TI_GETINFO	Get the TI protocol specific information from the transport protocol provider. The message issued to the TI_GETINFO *ioctl* is equivalent to the TI message type T_INFO_REQ and

	the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_INFO_ACK.
TI_OPTMGMT	Get, set, or negotiate protocol specific options with the transport protocol provider. The message issued to the TI_OPTMGMT *ioctl* is equivalent to the TI message type T_OPTMGMT_REQ, and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_OPTMGMT_ACK.

FILES

<sys/timod.h>
<sys/tiuser.h>
<sys/tihdr.h>
<sys/errno.h>

SEE ALSO

tirdwr(7).
STREAMS Primer.
STREAMS Programmer's Guide.
Network Programmer's Guide.

DIAGNOSTICS

If the *ioctl* system call returns with a value greater than 0, the lower 8 bits of the return value will be one of the TI error codes as defined in *<sys/tiuser.h>*. If the TI error is of type TSYSERR, then the next 8 bits of the return value will contain an error as defined in *<sys/errno.h>* [see *intro*(2)].

NAME

tirdwr – Transport Interface read/write interface STREAMS module

DESCRIPTION

tirdwr is a STREAMS module that provides an alternate interface to a transport provider which supports the Transport Interface (TI) functions of the Network Services library (see Section 3N). This alternate interface allows a user to communicate with the transport protocol provider using the *read*(2) and *write*(2) system calls. The *putmsg*(2) and *getmsg*(2) system calls may also be used. However, *putmsg* and *getmsg* can only transfer data messages between user and *stream*.

The *tirdwr* module must only be pushed [see I_PUSH in *streamio*(7)] onto a *stream* terminated by a transport protocol provider which supports the TI. After the *tirdwr* module has been pushed onto a *stream*, none of the Transport Interface functions can be used. Subsequent calls to TI functions will cause an error on the *stream*. Once the error is detected, subsequent system calls on the *stream* will return an error with *errno* set to EPROTO.

The following are the actions taken by the *tirdwr* module when pushed on the *stream*, popped [see I_POP in *streamio*(7)] off the *stream*, or when data passes through it.

push – When the module is pushed onto a *stream*, it will check any existing data destined for the user to ensure that only regular data messages are present. It will ignore any messages on the *stream* that relate to process management, such as messages that generate signals to the user processes associated with the *stream*. If any other messages are present, the I_PUSH will return an error with *errno* set to EPROTO.

write – The module will take the following actions on data that originated from a *write* system call:

- All messages with the exception of messages that contain control portions (see the *putmsg* and *getmsg* system calls) will be transparently passed onto the module's downstream neighbor.
- Any zero length data messages will be freed by the module and they will not be passed onto the module's downstream neighbor.
- Any messages with control portions will generate an error, and any further system calls associated with the *stream* will fail with *errno* set to EPROTO.

read – The module will take the following actions on data that originated from the transport protocol provider:

- All messages with the exception of those that contain control portions (see the *putmsg* and *getmsg* system calls) will be transparently passed onto the module's upstream neighbor.
- The action taken on messages with control portions will be as follows:

    + Messages that represent expedited data will generate an error. All further system calls associated with the *stream* will fail with *errno* set to EPROTO.

    + Any data messages with control portions will have the control portions removed from the message prior to passing the message on to the upstream neighbor.

    + Messages that represent an orderly release indication from the transport provider will generate a zero length data message, indicating the end of file, which will be sent to the reader of the *stream*. The orderly release message itself will be freed by the module.

    + Messages that represent an abortive disconnect indication from the transport provider will cause all further *write* and *putmsg* system calls to fail with *errno* set to ENXIO. All further *read* and *getmsg* system calls will return zero length data (indicating end of file) once all previous data has been read.

    + With the exception of the above rules, all other messages with control portions will generate an error and all further system calls associated with the *stream* will fail with *errno* set to EPROTO.

  - Any zero length data messages will be freed by the module and they will not be passed onto the module's upstream neighbor.

pop – When the module is popped off the *stream* or the *stream* is closed, the module will take the following action:

  - If an orderly release indication has been previously received, then an orderly release request will be sent to the remote side of the transport connection.

SEE ALSO

streamio(7), timod(7).
intro(2), getmsg(2), putmsg(2), read(2), write(2), intro(3) in the *Programmer's Reference Manual.*
STREAMS Primer.
STREAMS Programmer's Guide.
Network Programmer's Guide.

NAME

tp4 - Intel ISO TC4 compatible TLI network device driver

DESCRIPTION

The files */dev/tp4-??* provide an interface between applications which use the **Transport Library Interface** as specified in the **Network Programmer's Guide** and the *Intel SXM552/552A* networking board running *iNA961 Release 2.0 firmware.* The Intel board provides ISO compatible networking services for layers 1-4 including Transport Class 4 service. The *tp4* devices map this service to the Transport Library Interface.

The *tp4* devices accept network addresses in three formats: SubnetId, AreaId, and Release 1 Compatibility. These addressing formats are discussed in the **iNA960/iNA961 Programmer's Reference Manual,** Chapter 5.5. The addresses take the following forms for the *tp4* devices:

SubnetId

```
struct {
 unsigned char   net_addr_len;        /* either 0xa or 0xb */
 unsigned char   net_afi;             /* always 0x49 */
 unsigned short  net_subnetno;        /* dest. subnet id */
 unsigned char   net_subnetaddr[7];   /* dest. host id, 0xfe */
 unsigned char   net_nsap_id;         /* optional - dest. nsap */
 unsigned char   tsap_len;            /* always 0x2 */
 unsigned short  tsap;                /* dest. tsap */
 }
```

AreaId

```
struct {
 unsigned char   net_addr_len;        /* either 0xe or 0xf */
 unsigned char   net_afi;             /* always 0x49 */
 unsigned char   net_areaid[5];       /* dest. area id */
 unsigned char   net_subnetno;        /* dest. subnet id */
 unsigned char   net_subnetaddr[7];   /* dest. host id, 0xfe */
 unsigned char   net_nsap_id;         /* optional - dest. nsap */
 unsigned char   tsap_len;            /* always 0x2 */
 unsigned short  tsap;                /* dest. tsap */
 }
```

Compatibility

```
struct {
 unsigned char   net_addr_len;        /* always 0xc */
 unsigned char   net_subnetno[4];     /* always 0x1 */
 unsigned char   net_subnetaddr[6];   /* dest. host id */
 unsigned short  net_nsap_id;         /* always 0x1 */
 unsigned char   tsap_len;            /* always 0x2 */
 unsigned short  tsap;                /* dest. tsap */
 }
```

The minor device 0 is reserved for administrative operations. Any writes to this device will be taken as download requests for the board. Some ioctls can only be done to this device (see below).

The device */dev/iso-tp4* is provided for use by most applications. This name refers to the clone device entry for tp4 and will open any available minor device number. See the Streams Programmer's Guide for more information on clone opens.

The following *ioctl(2)* system calls are available:

Statistics

command = 1, arg must be a pointer to an array of integers
Returns the parameters specified in the **i552_stat** array defined in *sys/i552user.h.*

Reset

command = 3, arg = ignored
Resets the board attached to the fildes; fildes must be the result of opening minor device number 0.

FILES

/dev/iso-tp4, /dev/tp4-??

BUGS

This release of the iNA961 software is incompatible with previous releases. The compatibility mode addressing allows application software compatibility only.

There is no simple way to determine the board's hardware IEEE802 address other than at machine boot.

t_bind(3N) will return addresses only in the SubnetId format.

SEE ALSO

STREAMS Programmer's Guide.

Network Programmer's Guide.

Intel iNA960/iNA961 Programmer's Reference Manual.

NAME

tty – controlling terminal interface

DESCRIPTION

The file */dev/tty* is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

FILES

/dev/tty
/dev/tty*

SEE ALSO

console(7).

NAME

xt – multiplexed tty driver for AT&T windowing terminals

DESCRIPTION

The *xt* driver provides virtual *tty*(7) circuits multiplexed onto real *tty*(7) lines. It interposes its own channel multiplexing protocol as a line discipline between the real device driver and the standard *tty*(7) line disciplines.

Virtual *tty*(7) circuits are named by character-special files of the form */dev/xt???*. File names end in three digits, where the first two represent the channel group and the last represents the virtual *tty*(7) number (0-7) of the channel group. Allocation of a new channel group is done dynamically by attempting to open a name ending in **0** with the **O_EXCL** flag set. After a successful open, the *tty*(7) file onto which the channels are to be multiplexed should be passed to *xt* via the **XTIOCLINK** *ioctl*(2) request. Afterwards, all the channels in the group will behave as normal *tty*(7) files, with data passed in packets via the real *tty*(7) line.

The *xt* driver implements the protocol described in *xtproto*(5) and in *layers*(5). Packets are formatted as described in *xtproto*(5), while the contents of packets conform to the description in *layers*(5).

There are three groups of *ioctl*(2) requests recognized by *xt*. The first group contains all the normal tty *ioctl*(2) requests described in *termio*(7), with the addition of the following:

TIOCEXCL	Set exclusive use mode; no further opens are permitted until the file has been closed.
TIOCNXCL	Reset exclusive use mode; further opens are once again permitted.

The second group of *ioctl*(2) requests concerns control of the windowing terminal, and is described in the header file **<sys/jioctl.h>**. The requests are as follows:

JTYPE, JMPX	Both return the value **JMPX** . These are used to identify a terminal device as an *xt* channel.
JBOOT, JTERM	Both generate an appropriate command packet to the windowing terminal affecting the layer associated with the file descriptor argument to *ioctl*(2). They may return the error code **EIO** if the system *clist* is empty.
JTIMO, JTIMOM	**JTIMO** specifies the timeouts in seconds, and **JTIMOM** in milliseconds. Invalid except on channel 0. They may return the error code **EIO** if the system *clist* is empty.
JWINSIZE	Requires the address of a *jwinsize* structure as an argument. The window sizes of the layer associated with the file descriptor argument to *ioctl*(2) are copied to the structure.
JZOMBOOT	Generate a command packet to the windowing terminal to enter download mode on the channel associated with the file descriptor argument to *ioctl*(2), like **JBOOT**; but when the download is finished, make the layer a zombie

(ready for debugging). It may return the error code **EIO** if the system *clist* is empty.

JAGENT Send the supplied data as a command packet to invoke a windowing terminal agent routine, and return the terminal's response to the calling process. Invalid except on the file descriptor for channel 0. See *jagent*(5). It may return the error code **EIO** if the system *clist* is empty.

The third group of *ioctl*(2) requests concerns the configuration of *xt*, and is described in the header file **<sys/xt.h>**. The requests are as follows:

XTIOCTYPE Returns the value **XTIOCTYPE.**

XTIOCLINK Requires an argument that is a structure, *xtioclm*, containing a file descriptor for the file to be multiplexed and the maximum number of channels allowed. Invalid except on channel 0. This request may return one of the following errors:

EINVAL	*nchans* has an illegal value.
ENOTTY	*fd* does not describe a real *tty*(7) device.
ENXIO	*linesw* is not configured with *xt*.
EBUSY	An **XTIOCLINK** request has already been issued for the channel group.
ENOMEM	There is no system memory available for allocating to the *tty*(7) structures.
EIO	The **JTIMOM** packet described above could not be delivered.

HXTIOCLINK Like **XTIOCLINK,** but specifies that ENCODING MODE be used.

XTIOCTRACE Requires the address of a *Tbuf* structure as an argument. The structure is filled with the contents of the driver trace buffer. Tracing is enabled. This request is invalid if tracing is not configured.

XTIOCNOTRACE Tracing is disabled. This request is invalid if tracing is not configured.

XTIOCSTATS Requires an argument that is the address of an array of size S_NSTATS, of type *Stats_t*. The array is filled with the contents of the driver statistics array. This request is invalid if statistics are not configured.

XTIOCDATA Requires the address of a maximum-sized *Link* structure as an argument. The structure is filled with the contents of the driver *Link* data. This request is invalid if data extraction is not configured.

FILES

/dev/xt/??[0-7]	multiplexed special files
/usr/include/sys/jioctl.h	packet command types
/usr/include/sys/xtproto.h	channel multiplexing protocol definitions
/usr/include/sys/xt.h	driver specific definitions

SEE ALSO

layers(1), termio(7), tty(7).

ioctl(2), open(2), libwindows(3X), jagent(5), layers(5) in the *Programmer's Reference Manual*.

Section 8

Section 8

NAME

intro – introduction to system maintenance procedures

DESCRIPTION

This section outlines certain procedures that will be of interest to those charged with the task of system maintenance. Included are discussions of such topics as boot procedures, recovery from crashes, file backups, etc.

SEE ALSO

Operations/System Administration Guide.

NAME

sysdump – boot option to dump system memory image to floppy disk(s)

SYNOPSIS

sysdump

DESCRIPTION

The *sysdump* command dumps the system memory image to one or more floppy disks depending on the size of memory and user request. This memory image can later be analyzed by *crash*(1M). *sysdump* is invoked as a boot option.

When booted, *sysdump* begins an interactive procedure that prompts the user to insert the floppies to be loaded. The user has the option of quitting the session any time. This allows only the portion of the system image needed to be dumped.

The output of *sysdump* provides one input to *crash(1)*. The other input is the text file that was used to boot this system image. This is needed to provide symbolic reference to the system dump. The text file must be manually saved after the machine has been booted. If **/unix** was booted then this should be dumped to floppy to accompany the system dump.

FILES

/dev/rdsk/f0d9dt–Normal density (360 kbytes) floppy device
/dev/rdisk/f0q15dt–High density (1.2 Mbytes) floppy device
/dev/rmt0–Cartridge device
/unix -- the text file typically used to boot the machine
All of these devices may not be provided for every machine.

SEE ALSO

crash(1M).

DIAGNOSTICS

If a floppy diskette is inserted out of sequence, a message is printed. The user is allowed to insert a new diskette and continue the session.

WARNINGS

It is critical to provide access to the text file used to boot the machine. This file must be saved.

The diskettes should be labeled clearly so they can be loaded in the proper sequence.

The *sysdump*(8) command is part of the kernel debugger. It does not work without linking the kernel debugger to the UNIX system kernel.

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES