NCR

User Reference Manual

D1-0398-C
October, 1986

16-Bit Release 3.01
32-Bit Release 1.03

It is the policy of NCR Corporation to improve products as new technology, components, software, and firmware become available. NCR Corporation, therefore, reserves the right to change specifications without prior notice.

All features, functions, and operations described herein may not be marketed by NCR in all parts of the world. In some instances, photographs are of equipment prototypes. Therefore, before using this document, consult your NCR representative or NCR office for information that is applicable and current.

UNIX is a trademark of AT&T Bell Laboratories

To maintain the quality of our publications, we need your comments on the accuracy, clarity, organization, and value of this book.

Address correspondence to:

Technical Education/Publications
NCR Corporation
3325 Platt Springs Road
West Columbia, SC 29169
U.S.A.

UPDATE NOTIFICATION

Document No.: D1-0398-C

Date of Update: October, 1986

Document Title: TOWER User Reference Manual

Original Issue Date: November, 1985

REASON FOR UPDATE:

To include new release of software.

UPDATE INSTRUCTIONS:

The enclosed book is for TOWER 32 Operating System Release 1.03 and TOWER XP Operating System Release 3.01.

If you have TOWER 32 Operating System Release 1.02, keep your current book until you upgrade to Release 1.03.

If you have TOWER 32 Operating System Release 1.02 but do not have the Release 1.02 book, this book should be used. When using this book, be aware that the following manual pages only pertain to Release 3.01:

- ati ..... read and write ANSI format tapes
- backup ... backup, restore - backup or restore selected files
- packsf ... compress and uncompress sparse file
- pcdsk .... PC-DOS to UNIX file transfer
- tpcvt .... filter for old streaming tape format

If you have TOWER XP Operating System Release 3.01, replace the old book in its entirety with this new book.

If you have TOWER XP Operating System Release 3.00, keep your Release 3.00 book.

UPDATE NOTIFICATION

Document No.: D1-0398-C

Date of Update: October, 1986

Document Title: TOWER User Reference Manual

Original Issue Date: November, 1985

REASON FOR UPDATE:

To include new release of software.

UPDATE INSTRUCTIONS:

The enclosed book is for TOWER 32 Operating System Release 1.03 and TOWER XP Operating System Release 3.01.

If you have TOWER 32 Operating System Release 1.02, keep your current book until you upgrade to Release 1.03.

If you have TOWER 32 Operating System Release 1.02 but do not have the Release 1.02 book, this book should be used. When using this book, be aware that the following manual pages only pertain to Release 3.01:

- ati ..... read and write ANSI format tapes
- backup ... backup, restore - backup or restore selected files
- packsf ... compress and uncompress sparse file
- pcdsk .... PC-DOS to UNIX file transfer
- tpcvt .... filter for old streaming tape format

If you have TOWER XP Operating System Release 3.01, replace the old book in its entirety with this new book.

If you have TOWER XP Operating System Release 3.00, keep your Release 3.00 book.

TABLE OF CONTENTS

1. Commands and Application Programs

6. Games

OVERVIEW

RELEASE: The information in this *User Reference Manual* applies to Release 3.01 of the 16-bit and Release 1.03 of the 32-bit operating systems based on UNIX.

AUDIENCE: The audiences for this book are general users, programmers, analysts, and system support personnel. It is expected that the user is familiar with UNIX, another operating system derived from UNIX, or appropriate operating system courses.

CONTENT: This manual describes the user commands of the operating system. For a description of the operating system calls, subroutines, file formats, and miscellaneous facilities, refer to the *Programmer Reference Manual.* For a description of the maintenance commands, special files, and maintenance procedures, refer to the *Superuser Reference Manual.*

The manual is divided into three sections, one containing inter-filed subclasses:

1. Commands and Application Programs:
   1. General-Purpose Commands
   1B. Berkeley Commands
   1C. Communication Commands
   1G. Graphics Commands
6. Games.

Section 1 (*Commands and Application Programs*) describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by user programs. Commands generally reside in the directory **/bin** (for binary programs). Some programs also reside in **/usr/bin** and **/usr/ucb**, to save space in **/bin**. These directories are searched automatically by the command interpreter called the *shell.*

Section 6 (*Games*) describes the games and educational programs that reside in the directory **/usr/games**.

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

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

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

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

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

- 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.
- Braces { } around an argument prototype indicate that at least one of the enclosed arguments must be chosen.
- Ellipses ... are used to show that the previous argument prototype may be repeated.
- A final convention is used by the commands themselves. An argument beginning with a minus −, plus +, or equal sign = is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with −, +, or =.

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

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

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

The **SEE ALSO** part gives pointers to related information. References of the form **name**(N), where N is the number 1 or 6 possibly followed by a letter, refer to entries in this manual. References of the form **name**(N), where N is a number 2 through 5 possibly followed by a letter, refer to entries in the *Programmer Reference Manual.* References of the form **name**(1M), **name**(7), or **name**(8) refer to entries in the *Superuser Reference Manual.*

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

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

The **RESTRICTIONS** part gives known restrictions and deficiencies. Occasionally, the suggested fix is also described.

The **SUPPORT STATUS** part specifies the item as supported or not supported. The operating system is fully supported. Included in the system, however, are unsupported items you might find useful. For example, SVS-FORTRAN, DI-3000 Graphics, and a line printer spooler are supported for the system. The UNIX FORTRAN, graphics, and spooler items are included as unsupported items.

All entries are available online via the *man*(1) command, if these files are installed during software installation.

A table of contents, a permuted index derived from that table, and a module contents are included in this manual.

The *permuted index* is a combined index for the *User Reference Manual, Programmer Reference Manual,* and *Superuser Reference Manual.* On each *permuted index* line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

A *module contents,* defining the components of each operating system module, is also included. The operating system is distributed as several modules any or all of which may be installed on your system. Before

using this manual, check with your system administrator to determine which modules are installed on your system. The *module contents* is a combined list for entries in the *User Reference Manual*, *Programmer Reference Manual*, and *Superuser Reference Manual*.

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 Guide* for a more complete introduction to the system.

Logging in. You must call the UNIX system from an appropriate terminal. The UNIX system supports full-duplex ASCII terminals. You must also have a valid user name, which may be obtained together with the telephone number(s) of your UNIX system from the administrator of your system. On some UNIX systems, there are separate telephone numbers for each available terminal speed, while on other systems several speeds may be served by a single telephone number. In the latter case, there is one preferred speed; if you dial in from a terminal set to a different speed, you are greeted by a string of meaningless characters (the **login:** message at the wrong speed). Keep pressing break or control-break until the **login:** message appears. Hard-wired terminals usually are set to the correct speed.

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

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

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

Logging out. There are two ways to log out. 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 terminates and the **login:** message appears again.

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

The 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, the output has interspersed in it the input characters. However, whatever you type is

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. When the read-ahead limit is exceeded, the system discards *all* the saved characters.

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

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

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

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

Besides adapting to the speed of the terminal, 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. If you get into the wrong mode, the *stty*(1) command can rescue you.

Tab characters are used freely in the 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 sets or resets this mode. The system assumes that tabs are set every eight character positions. The *tabs*(1) command sets 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 listening to your terminal. The shell reads the lines you type, splits them into a command name and its arguments, and executes the command. A command is simply an executable program. Normally, the shell looks first in your current directory (see *The current directory* below) for a program with the given name, and if none is there, then in system directories. There is nothing

special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and arrange for the shell to find them there.

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

When a program terminates, the shell ordinarily regains control and types a $ to the terminal 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 file system is arranged in a hierarchy of directories. When the system administrator gave you a user name, he also created a directory for you (ordinarily with the same name as your user name, and known as your *login* or *home* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name you type is by default assumed to be in that directory. Because you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. Permissions to have access to other directories and files have been granted or denied to you by their respective owners or by the system administrator. To change the current directory use *cd*(1).

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

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

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

For a fuller discussion of the file system, see the *System Operation* book. It may also be useful to glance through Section 2 of the *Programmer Reference Manual* which discusses system calls even if you do not intend to use the system at that level.

Writing a program. To enter the text of a source program into a UNIX system file, use *ed*(1), *edit*(1), *ex*(1), or *vi*(1). After the program text has been entered with the editor and written into a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. The resulting program can be run by giving its name to the shell in response to the $ prompt.

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

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

Inter-user Communication. Certain commands provide *inter-user* communication. To communicate with another user currently logged in, *write*(1) is used; *mail*(1) leaves a message whose presence is announced to another user when he next logs in. The corresponding entries in this manual also suggest how to respond to these two commands.

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

PERMUTED INDEX

MODULE CONTENTS

The operating system is distributed as a base module and several functionally independent modules. Each independent module provides a specific capability. This *module contents* defines the content of each of the modules of the operating system. Following a short description of each module, a combined contents for entries in the *User Reference Manual*, *Programmer Reference Manual*, and *Superuser Reference Manual* specifies the operating system module which contains the feature (na specifies not applicable). Before using this manual, check with your system administrator to determine which operating system modules are installed on your system.

MODULE DESCRIPTIONS

Business Base Module (bbase)
The *Business Base Module (bbase)* is the minimum operating system. This module contains the basic operating system utilities, menu programs for the system administrator (sa login) and end users, and reconfiguration and maintenance features for the support technician (ncrm login).

Extension Module (exten)
The *Extension Module (exten)* contains non-essential operating system utilities such as *vi*, *spell*, and *csh* that may be useful to some users.

Miscellaneous Module (misc)
The *Miscellaneous Module (misc)* contains further non-essential operating system utilities such as games and the UNIX line printer spooler facility.

Graphics Module (graph)
The *Graphics Module (graph)* contains all UNIX graphics facilities as unsupported software.

System Documentation Module (man)
The *System Documentation Module (man)* includes the online manual pages, related utilities, and the system administrator help files.

Software Development Module (devel)
The *Software Development Module (devel)* includes the facilities for software development such as C language tools, *f77*, and the Source Code Control System (SCCS) facility.

68010 Compiler Module (compile)
The *68010 Compiler Module (compile)* is available only for 32-bit systems and includes the 16-bit C compiler and its associated libraries and routines. On 16-bit systems, this software is in the *devel* module.

System Accounting Module (acct)
The *System Accounting Module (acct)* provides the complete system accounting facility: methods to collect per-process resource utilization data, record connect sessions, monitor disk utilization, and charge fees

hp	handle special functions of HP 2640 and 2621-series terminals	graph
id	print user and group IDs and names	bbase
ipcrm	remove a message queue, semaphore set or shared memory id	bbase
ipcs	report inter-process communication facilities status	bbase
join	relational database operator	exten
kill	terminate a process	bbase
last	indicate last logins of users and teletypes	bbase
ld	link editor for common object files	bbase
lex	generate programs for simple lexical tasks	devel
line	read one line	bbase
lint	a C program checker	devel
login	sign on	bbase
logname	get login name	bbase
look	find lines in a sorted list	exten
lorder	find ordering relation for an object library	devel
lp	send requests to an LP line printer	misc
lpstat	print LP status information	misc
ls	list contents of directory	bbase
ls	list contents of directory (Berkeley)	bbase
m4	macro processor	devel
machid	provide truth value about your processor type	bbase
mail	send mail to users or read mail	bbase
mailx	interactive message processing system	bbase
make	maintain, update, and regenerate groups of programs	bbase
makekey	generate encryption key	exten
man	print entries in this manual	man
mesg	permit or deny messages	bbase
mkdir	make a directory	bbase
mklost+found	make a lost+found directory for fsck	bbase
mkstr	create an error message file by massaging C source	devel
mm	print/check documents formatted with the MM macros	man
mmt	typeset documents, viewgraphs, and slides	man
more	file perusal filter for crt viewing	bbase
newform	change the format of a text file	exten
newgrp	log in to a new group	bbase
news	print news items	bbase
nice	run a command at low priority	bbase
nl	line numbering filter	exten
nm	print name list of common object file	bbase
nohup	run a command immune to hangups and quits	bbase
nroff	format or typeset text	man
od	octal dump	bbase
pack	compress files	exten
packsf	compress and uncompress sparse file	exten
passwd	change login password	bbase
paste	merge same lines of several files or subsequent lines of one file	exten
pg	file perusal filter for soft-copy terminals	bbase
pr	print files	bbase
print	line printer spooler	bbase

prof	display profile data	devel
prs	print an SCCS file	devel
ps	report process status	bbase
ptx	permuted index	man
pwd	working directory name	bbase
ratfor	rational Fortran dialect	devel
regcmp	regular expression compile	devel
rev	reverse lines of a file	bbase
rm	remove files or directories	bbase
rmdel	remove a delta from an SCCS file	devel
sact	print current SCCS file editing activity	devel
sag	system activity graph	acct
sar	system activity reporter	acct
sccsdiff	compare two versions of an SCCS file	devel
sdb	symbolic debugger	devel
sdiff	side-by-side difference program	exten
sed	stream editor	bbase
sh	shell, the standard/restricted command programming language	bbase
shl	shell layer manager	exten
size	print section sizes of common object files	devel
sleep	suspend execution for an interval	bbase
sln	link files symbolically	bbase
sno	SNOBOL interpreter	misc
sort	sort and/or merge files	bbase
spell	find spelling errors	exten
spline	interpolate smooth curve	graph
split	split a file into pieces	bbase
spool	spool queue manager	bbase
ssp	make output single spaced	bbase
stat	statistical network useful with graphical commands	graph
strings	find the printable strings in a object, or other binary, file	devel
strip	strip symbol and line number information from object file	devel
stty	set the options for a terminal	bbase
su	become superuser or another user	bbase
sum	print checksum and block count of a file	bbase
sync	update the super block	bbase
tabs	set tabs on a terminal	misc
tail	deliver the last part of a file	bbase
tar	tape file archiver	bbase
tbl	format tables for nroff or troff	man
tc	phototypesetter simulator	misc
tee	copy input to standard output and to files	bbase
test	condition evaluation command	bbase
time	time a command	bbase
timex	time a command; report process data and system activity	acct
toc	graphical table of contents routines	graph
touch	update access and modification times of a file	devel
tpcvt	filter for old streaming tape format	misc

tplot	graphics filters	graph
tput	query terminfo database	bbase
tr	translate characters	bbase
true	provide truth values	bbase
tsort	topological sort	exten
tty	get the name of the terminal	bbase
ul	underline output for a terminal	bbase
umask	set file-creation mode mask	bbase
uname	print name of current UNIX system	bbase
unget	undo a previous get of an SCCS file	devel
uniq	report repeated lines in a file	exten
units	interactive conversion program	exten
uucp	UNIX system to UNIX system copy	bbase
uustat	uucp status inquiry and job control	bbase
uuto	public UNIX-to-UNIX system file copy	bbase
uux	UNIX-to-UNIX system command execution	bbase
val	validate SCCS file	devel
vc	version control	devel
vi	screen-oriented (visual) display editor based on ex	exten
wait	await completion of process	bbase
wc	word count	bbase
what	identify SCCS files	bbase
whereis	locate source, binary, and or manual for program	man
who	who is on the system	bbase
write	write to another user	bbase
xargs	construct argument list(s) and execute command	bbase
xstr	extract strings from C programs to implement shared strings	devel
yacc	yet another compiler-compiler	devel

6. Games

arithmetic	provide drill in arithmetic problems	misc
back	the game of backgammon	misc
bj	the game of black jack	misc
craps	the game of craps	misc
maze	generate a maze	misc
moo	guessing game	misc
ttt	tic-tac-toe	misc
wump	the game of hunt-the-wumpus	misc

PROGRAMMER REFERENCE MANUAL CONTENTS

2. System Calls

access	determine accessibility of a file	devel
acct	enable or disable process accounting	devel
alarm	set the process alarm clock for a process	devel
brk	change data segment space allocation	devel
chdir	change working directory	devel
chmod	change mode of file	devel
chown	change owner and group of a file	devel
chroot	change root directory	devel

close	close a file descriptor	devel
creat	create a new file or rewrite an existing one	devel
dup	duplicate an open file descriptor	devel
exec	execute a file	devel
exit	terminate process	devel
fcntl	file control	devel
ffp	floating point processor access	devel
fork	create a new process	devel
getpid	get process, process group, and parent process IDs	devel
getuid	get real user, effective user, real group, and effective group IDs	devel
ioctl	control device	devel
kill	send a signal to a process or a group of processes	devel
link	link to a file	devel
lockf	provide exclusive file regions for reading and writing	devel
lseek	move read/write file pointer	devel
mknod	make a directory, or a special or ordinary file	devel
mount	mount a file system	devel
msgctl	message control operations	devel
msgget	get message queue	devel
msgop	message operations	devel
nice	change priority of a process	devel
open	open for reading or writing	devel
pause	suspend process until signal	devel
pipe	create an interprocess channel	devel
plock	lock process, text, or data in memory	devel
profil	execution time profile	devel
ptrace	process trace	devel
pwrnote	power recovery notification	devel
pwrtime	power recovery interval to single-user state	devel
read	read from file	devel
rmnt	mount and unmount directorys across file systems	devel
semctl	semaphore control operations	devel
semget	get a set of semaphores	devel
semop	semaphore operations	devel
sernum	get serial number of current system	devel
setpgrp	set process group ID	devel
setuid	set user and group IDs	devel
shmctl	shared memory control operations	devel
shmget	get shared memory segment	devel
shmop	shared memory operations	devel
signal	specify what to do upon receipt of a signal	devel
slink	link files across file systems	devel
stat	get file status	devel
stime	set time	devel
swrite	synchronously write on a file	devel
sync	update super-block	devel
time	get time	devel
times	get process and child process times	devel
tslice	set/get time slice	devel
ulimit	get and set user limits	devel
umask	set and get file creation mask	devel

umount	unmount a file system	devel
uname	get name of current UNIX system	devel
unlink	remove directory entry	devel
ustat	get file system statistics	devel
utime	set file access and modification times	devel
wait	wait for child process to stop or terminate	devel
write	write on a file	devel

3. Subroutines

a64l	convert between long integer and base-64 ASCII string	devel
abort	generate an IOT fault	devel
abort	terminate Fortran program	devel
abs	return integer absolute value	devel
abs	Fortran absolute value	devel
aimag	Fortran imaginary part of complex argument	devel
aint	Fortran integer part intrinsic function	devel
assert	verify program assertion	devel
bessel	Bessel functions	devel
bool	Fortran bitwise boolean functions	devel
bsearch	binary search	devel
clock	report CPU time used	devel
conjg	Fortran complex conjugate intrinsic function	devel
conv	translate characters	devel
crypt	generate DES encryption	devel
ctermid	generate file name for terminal	devel
ctime	convert date and time to string	devel
ctype	classify characters	devel
curses	screen functions with optimal cursor motion	devel
curses	CRT screen handling and optimization package	devel
cuserid	get character login name of the user	devel
dial	establish an out-going terminal line connection	devel
dim	positive difference intrinsic functions	devel
dprod	double precision product intrinsic function	devel
drand48	generate uniformly distributed pseudo-random numbers	devel
ecvt	convert floating-point number to string	devel
end	last locations in program	devel
erf	error function and complementary error function	devel
exp	Fortran exponential, logarithm, square root intrinsic functions	devel
exp	exponential, logarithm, power, square root functions	devel
fclose	close or flush a stream	devel
ferror	stream status inquiries	devel
flcvt	float format conversions	devel
floor	floor, ceiling, remainder, absolute value functions	devel
fopen	open a stream	devel
fread	binary input/output	devel
frexp	manipulate parts of floating-point numbers	devel
fseek	reposition a file pointer in a stream	devel
ftw	walk a file tree	devel
ftype	explicit Fortran type conversion	devel

gamma	log gamma function	devel
getarg	return Fortran command-line argument	devel
getc	get character or word from stream	devel
getcwd	get path-name of current working directory	devel
getenv	return value for environment name	devel
getenv	return Fortran environment variable	devel
getgrent	get group file	devel
getlogin	get login name	devel
getopt	get option letter from argument vector	devel
getpass	read a password	devel
getpw	get name from UID	devel
getpwent	get password	devel
gets	get a string from a stream	devel
getut	access utmp file entry	devel
hsearch	manage hash search tables	devel
hypot	Euclidean distance function	devel
iargc	number of command line arguments	devel
index	return location of Fortran substring	devel
l3tol	convert between 3-byte integers and long integers	devel
ldahread	read the archive header of a member of an archive file	devel
ldclose	close a common object file	devel
ldfhread	read the file header of a common object file	devel
ldgetname	retrieve symbol name for common object file symbol table entry	devel
ldlread	manipulate line number entries of a common object file function	devel
ldlseek	seek to line number entries of a section of a common object file	devel
ldohseek	seek to the optional file header of a common object file	devel
ldopen	open a common object file for reading	devel
ldrseek	seek to relocation entries of a section of a common object file	devel
ldshread	read an indexed/named section header of a common object file	devel
ldsseek	seek to an indexed/named section of a common object file	devel
ldtbindex	compute index of symbol table entry of object file	devel
ldtbread	read an indexed symbol table entry of a common object file	devel
ldtbseek	seek to the symbol table of a common object file	devel
len	return length of Fortran string	devel
lockf	record locking on files	devel
logname	return login name of user	devel
lsearch	linear search and update	devel
malloc	main memory allocator	devel
malloc	fast main memory allocator	devel
matherr	error-handling function	devel
max	Fortran maximum-value functions	devel
mclock	return Fortran time accounting	devel
memory	memory operations	devel
min	Fortran minimum-value functions	devel
mktemp	make a unique file name	devel

mod	Fortran remaindering intrinsic functions	devel
monitor	prepare execution profile	devel
nlist	get entries from name list	devel
perror	system error messages	devel
plot	graphics interface subroutines	graph
popen	initiate pipe to/from a process	devel
printf	print formatted output	devel
putc	put character or word on a stream	devel
putenv	change or add value to environment	devel
putpwent	write password file entry	devel
puts	put a string on a stream	devel
qsort	quicker sort	devel
rand	simple random-number generator	devel
rand	Fortran uniform random-number generator	devel
regcmp	compile and execute regular expression	devel
round	Fortran nearest integer functions	devel
scanf	convert formatted input	devel
setbuf	assign buffering to a stream	devel
setjmp	non-local goto	devel
sign	Fortran transfer-of-sign intrinsic function	devel
signal	specify Fortran action on receipt of a system signal	devel
sleep	suspend execution for interval	devel
sputl	access long numeric data in a machine independent fashion.	devel
ssignal	software signals	devel
stdio	standard buffered input/output package	devel
stdipc	standard interprocess communication package	devel
strcmp	string comparision intrinsic functions	devel
string	string operations	devel
strtod	convert string to double-precision number	devel
strtol	convert string to integer	devel
swab	swap bytes	devel
system	issue a shell command from Fortran	devel
system	issue a shell command	devel
termcap	terminal independent operation routines	devel
tmpfile	create a temporary file	devel
tmpnam	create a name for a temporary file	devel
trig	Fortran trigonometric intrinsic functions	devel
trig	trigonometric functions	devel
trigh	Fortran hyperbolic intrinsic functions	devel
trigh	hyperbolic functions	devel
tsearch	manage binary search trees	devel
ttyname	find name of a terminal	devel
ttyslot	find the slot in the utmp file of the current user	devel
ungetc	push character back into input stream	devel
vprintf	print formatted output of a varargs argument list	devel
vprintf	print formatted output of a varargs argument list	devel

4. File Formats

a.out	common assembler and link editor output	na
acct	per-process accounting file format	na
alert	error records for devices exceeding threshold values	na

alertmesg	logalert summary message file	bbase
ar	common archive file format	na
checklist	list of file systems processed by fsck	na
core	format of core image file	na
cpio	format of cpio archive	na
dir	format of directories	na
errfile	error-log file format	na
filehdr	file header for common object files	na
fs	format of system volume	na
fspec	format specification in text files	na
gettydefs	speed and terminal settings used by getty	bbase
gps	graphical primitive string, format of graphical files	na
group	group file	bbase
inittab	script for the init process	bbase
inode	format of an inode	na
issue	issue identification file	bbase
ldfcn	common object file access routines	na
linenum	line number entries in a common object file	na
master	master device information table	bbase
mnttab	mounted file system table	na
passwd	password file	bbase
plot	graphics interface	na
profile	setting up an environment at login time	na
reloc	relocation information for a common object file	na
rmnttab	mounted directory table	bbase
sccsfile	format of SCCS file	na
scnhdr	section header for a common object file	na
syms	common object file symbol table format	na
sysident	date and release number of the operating system	bbase
term	format of compiled term file	bbase
terminfo	terminal capability data base	bbase
threshold	logalert threshold file	bbase
utmp	utmp and wtmp entry formats	na

5. Miscellaneous Facilities

ascii	map of ASCII character set	man
btermcap	terminal capability data base	misc
environ	user environment	na
eqnchar	special character definitions for eqn and neqn	bbase
fcntl	file control options	devel
font	description files for device-independent troff	man
greek	graphics for the extended TTY-37 type-box	man
man	macros for formatting entries in this manual	man
math	math functions and constants	devel
me	macros for formatting papers	man
mm	the MM macro package for formatting documents	man
mosd	the OSDD adapter macro package for formatting documents	man
mptx	the macro package for formatting a permuted index	man
ms	text formatting macros	man
mv	troff macro package to typeset view graphs and slides	man
prof	profile within a function	devel

SUPERUSER REFERENCE MANUAL CONTENTS

1. Maintenance Commands and Application Programs

fsdb	file system debugger	bbase
fuser	identify processes using a file or file structure	exten
fwtmp	manipulate connect accounting records	acct
getty	set terminal type, modes, speed, and line discipline	bbase
icheck	display inode number	bbase
init	process control initialization	bbase
install	install commands	exten
killall	kill all active processes	bbase
ldhpsio	high performance serial	bbase
link	exercise link and unlink system calls	bbase
logalert	error log threshold analysis utility	bbase
lpadmin	configure the LP spooling system	misc
lpd	line printer daemon	bbase
lpsched	start/stop the LP request scheduler and move requests	misc
mkfs	construct a file system	bbase
mknod	build special file	bbase
mount	mount and dismount file system	bbase
mvdir	move a directory	bbase
ncheck	generate names from i-numbers	bbase
newfs	construct a file system	bbase
profiler	operating system profiler	devel
pwck	password/group file checkers	bbase
rmount	mount and unmount directorys across file systems	bbase
runacct	run daily accounting	acct
sadp	disk access profiler	acct
sar	system activity report package	acct
setmnt	establish mount table	bbase
setrmnt	establish rmount table	bbase
shutdown	shutdown - shutdown system	bbase
single	single - go from multi-user to single-user mode	bbase
skyload	load the SKY Floating Point Processor	misc
spooldev	spool system device table manager	bbase
sublock	display contents of superblock from	bbase
tic	terminfo compiler	bbase
uuclean	uucp spool directory clean-up	bbase
uusub	monitor uucp network	comm
verchk	display SCCS version number and file attributes	comm
verify	turn on/off read after write check for device	bbase
volcopy	copy file systems with label checking	bbase
wall	write to all users	bbase
whodo	who is doing what	bbase

7. Special Files

err	error-logging interface	bbase
hpsio	high performance serial	bbase
ios	intelligent 8-channel serial	bbase
lp	line printer	bbase
mem	core memory	bbase
nec	nec —	bbase
null	the null file	bbase
prf	operating system profiler	bbase
sd	driver for the SCSI disks	bbase

ss	driver for the SCSI tapes	bbase
sxt	pseudo-device driver	devel
termio	general terminal interface	bbase
tp	driver for 5.25 and 8 inch streaming tapes	bbase
trace	event-tracing driver	devel
tty	controlling terminal interface	bbase
wd	driver for the 5.25 inch disks	bbase
xl	driver for the 8 inch Winchester disks	bbase

8. Maintenance

boot	startup procedure	na
diag	run in-service diagnostics	bbase
inserv	inservice diagnostics	bbase
l0diag	perform automatic level 0 diagnostics	na
rc	command script for system housekeeping	bbase
sus	startup procedure	na
suscmd	invoke Start-Up-Subsystem function	na
suslog	invoke Start-Up-Subsystem (SUS) log driver	na

NAME

intro — introduction to commands and application programs

DESCRIPTION

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

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

COMMAND SYNTAX

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

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

where:

name
The name of an executable file.

option
— *noargletter*(*s*) or,
— *argletter*<>*optarg*
where <> is optional white space.

noargletter
A single letter representing an option without an argument.

argletter
A single letter representing an option requiring an argument.

optarg
Argument (character string) satisfying preceding *argletter*.

cmdarg
Path name (or other command argument) *not* beginning with — or, — by itself indicating the standard input.

SEE ALSO

getopt(1), exit(2), wait(2), getopt(3C).
Section 6 of this manual for computer games.

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 system byte is 0 for normal termination; the program byte is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. The program byte is called either *exit code*, *exit status*, or *return code* and is described only where special conventions are involved.

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.

SUPPORT STATUS

Supported.

NAME

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

SYNOPSIS

300 [+12] [−n] [−dt,l,c]

300s [+12] [−n] [−dt,l,c]

DESCRIPTION

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

neqn file ... | nroff | 300

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

The behavior of *300* can be modified to handle 12-pitch text, fractional line spacings, messages, and delays using the options described below:

+12 permits use of 12-pitch, 6 lines/inch text. The user should turn the PITCH to 12. DASI 300 terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch.

−n controls the size of half-line spacing. A half-line is normally equal to 4 vertical plot increments. Since 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 for halfline spacing so that the appearance of subscripts and superscripts can be altered. For example, *nroff* half-lines can be changed to quarter-lines by using −2. Using −3 and setting the PITCH switch to 12 generates appropriate half-lines for 12-pitch, 8 lines/inch mode.

−dt,l,c controls delay factors. 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. This option attempts to alleviate this problem.

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. A value of zero for *t* or *c* results in two null bytes per tab (character). If a line is longer than *l* bytes, 1+(total length)/20 nulls are inserted at the end of that line. 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 default setting for this option is **−d3,90,30**. Items omitted from the end of the list imply the use of the default value for those items. 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.

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

Greek(5) shows the *neqn* names of, and resulting output for, the Greek and special characters supported by *300*.

SEE ALSO

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

RESTRICTIONS

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 forms tractor 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.

SUPPORT STATUS

Not supported.

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.

OPTIONS

−t Eliminates waits between pages (useful for directing output into a file).

−n Starts printing at the current cursor position and never erases the screen.

−c*N*
Divides the screen into *N* columns and waits after the last column.

−p*L* Sets page length to *L*; *L* accepts the scale factors **i** (inches) and **l** (lines); default is lines. For example, **−p11i** sets page length to 11 inches; **−p11l** or **−p11** sets page length to 11 lines.

SEE ALSO

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

SUPPORT STATUS

Not supported.

NAME

450 − handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

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

neqn file ... | nroff | 450

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

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

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

nroff −T450 files ...

or

nroff −T450−12 files ...

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

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

SEE ALSO

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

RESTRICTIONS

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 forms tractor 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.

SUPPORT STATUS

Not supported.

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 following:

COMMAND NAME	− prepended with a # if executed with super-user privileges.
USER	−
TTYNAME	− a ? if a process is not associated with a known terminal type.
START TIME	−
END TIME	−
REAL (SEC)	−
CPU (SEC)	−
MEAN SIZE(K)	−
F	− (optional) fork/exec flag: 1 for fork without exec.
STAT	− (optional) the system exit status.
HOG FACTOR	−
KCORE MIN	−
CPU FACTOR	−
CHARS TRNSFD	−
BLOCKS /WD	− total blocks read and written.

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.

File arguments (if given) 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***?*.

OPTIONS

−a Show some average statistics about the process selected. The statistics are 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 *CPU "hog" time 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:

user ID	a login name converted to a user ID
#	designates only those processes executed with *super-user* privileges
?	designates 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; 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 the CPU time 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

ps(1), su(1), acct(2), acct(4), utmp(4), acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M).

RESTRICTIONS

Acctcom only reports 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.

SUPPORT STATUS

Not supported.

NAME

adb − absolute debugger

SYNOPSIS

adb [−w] [objfil [corfil]]

DESCRIPTION

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

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

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

In general requests to *adb* are of the form

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

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

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

EXPRESSIONS

. The value of *dot*.

\+ The value of *dot* incremented by the current increment.

^ The value of *dot* decremented by the current increment.

" The last *address* typed.

integer

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

integer.fraction

A 32-bit floating point number.

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

< name

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

... r5 sp pc ps.

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

_ symbol
In C, the *true name* of an external symbol begins with _. It may be necessary to utter this name to distinguish it from internal or hidden variables of a program.

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

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

Monadic operators:

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

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

−exp Integer negation.

~exp Bitwise complement.

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

e1+e2 Integer addition.

e1−e2 Integer subtraction.

*e1*e2* Integer multiplication.

e1%e2 Integer division.

e1&e2 Bitwise conjunction.

e1|e2 Bitwise disjunction.

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

COMMANDS

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

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

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

=f The value of *address* itself is printed in the styles indicated by the format *f*. (For i format ? is printed for the

parts of the instruction that reference subsequent words.)

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

o 2 Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
O 4 Print 4 bytes in octal.
q 2 Print in signed octal.
Q 4 Print long signed octal.
d 2 Print in decimal.
D 4 Print long decimal.
x 2 Print 2 bytes in hexadecimal.
X 4 Print 4 bytes in hexadecimal.
u 2 Print as an unsigned decimal number.
U 4 Print long unsigned decimal.
f 4 Print the 32 bit value as a floating point number.
F 8 Print double floating point.
b 1 Print the addressed byte in octal.
c 1 Print the addressed character.
C 1 Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.
s *n* Print the addressed characters until a zero character is reached.
S *n* Print a string using the @ escape convention. The value *n* is the length of the string including its zero terminator.
Y 4 Print 4 bytes in date format (see *ctime*(3C)).
i Print as 68000 instructions.
a 0 Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

/ local or global data symbol
? local or global text symbol
= local or global absolute symbol

p 2 Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.
t 0 When preceded by an integer, tabs to the next appropriate tab stop. For example, 8t moves to the next 8-space tab stop.
r 0 Print a space.
n 0 Print a new-line.
"..." 0
Print the enclosed string.
^ *Dot* is decremented by the current increment. Nothing is printed.

\+ *Dot* is incremented by 1. Nothing is printed.
− *Dot* is decremented by 1. Nothing is printed.

new-line
Repeat the previous command with a *count* of 1.

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

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

[?/]m *b1 e1 f1*[?/]
New values for (*b1, e1, f1*) are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the ? or / is followed by * then the second segment (*b2,e2,f2*) of the mapping is changed. If the list is terminated by ? or / then the file (*objfil* or *corfil*, respectively) is used for subsequent requests. (So that, for example, /m? causes / to refer to *objfil*.)

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

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

$*modifier*
Miscellaneous commands. The available *modifiers* are:

- <*f* Read commands from the file *f* and return.
- >*f* Send output to the file *f*, which is created if it does not exist.
- **r** Print the general registers and the instruction addressed by **pc**. *Dot* is set to **pc**.
- **b** Print all breakpoints and their associated counts and commands.
- **c** C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of **fp**). If **C** is used then the names and 16-bit values of all automatic and static variables are printed for each active function. If *count* is given then only the first *count* frames are printed.
- **e** The names and values of external variables are printed.
- **w** Set the page width for output to *address* (default 80).
- **s** Set the limit for symbol matches to *address* (default 255).
- **o** All integers input are regarded as octal.
- **d** Reset integer input as described in *EXPRESSIONS*.
- **q** Exit from *adb*.
- **v** Print all non-zero variables in octal.

m Print the address map.

:modifier

Manage a subprocess. Available modifiers are:

bc Set breakpoint at *address*. The breakpoint is executed *count*−1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command sets *dot* to zero then the breakpoint causes a stop.

d Delete breakpoint at *address*.

r Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. The value *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on when the subprocess is entered.

cs The subprocess is continued with signal *s* (see *signal*(2)). If *address* is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.

ss As for **c** except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfil* is run as a subprocess as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

k The current subprocess, if any, is terminated.

VARIABLES

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

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

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

b The base address of the data segment.
d The data segment size.
e The entry point.
f File offset to text segment data.
m The *magic* number (0407(0x107) or 0410(0x108)).
s The stack segment size.
t The text segment size.

ADDRESSES

The address in a file associated with a written address is

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

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

otherwise

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

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

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

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

"Adb" when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned non-zero status.

RESTRICTIONS

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

When single stepping, system calls do not count as an executed instruction.

Local variables whose names are the same as an external variable may make the accessing of the external incorrect.

The expression *routine.expression* is not supported, except for retrieving the stack frame address for an active routine.

The C stack backtrace does not display the call parameters.

SUPPORT STATUS

Supported.
Local variables are not supported.

NAME

admin − create and administer SCCS files

SYNOPSIS

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

DESCRIPTION

Admin is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which may appear in any order, consist of options, which begin with −, and named files (note that SCCS file names must begin with the characters s.). If a named file does not exist, it is created, and its parameters are initialized according to the specified options. Unitialized parameters assume a default value. If a named file does exist, parameters corresponding to specified options are changed, and other parameters are left as is.

If a directory is named, *admin* behaves as though each file in the directory were specified as a named file. If a name of − is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files are silently ignored.

OPTIONS

−n Indicates that a new SCCS file is to be created.

−i[*name*] Indicates that the input text for a new SCCS file is the file *name*. The input text constitutes the first delta of the file (see **−r** for delta numbering scheme). If *name*. is omitted, the input text is standard input, read until an end-of-file is reached. If **−i** is omitted, the SCCS file is created empty.

Note: only one SCCS file may be created when **−i** is used. Using a single *admin* to create two or more SCCS files requires that they be created empty (no **−i** option specified).

Also note that the **−i** option implies the **−n** option.

−r*rel* Indicates that the initial delta is inserted into release *rel*. **−r** must be used in conjunction with **−i**. If **−r** is omitted, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are name 1.1).

−t[*name*] Indicates the file *name* is the source of the descriptive text for the SCCS file. When creating a new SCCS file (that is, when used with the options **−n** and/or **−i**) *name* must be supplied. When modifying an existing SCCS file, the absence of *name* removes any descriptive text; otherwise, the contents of *name* replace any descriptive text currently in the SCCS file.

−f*flag* Specifies a *flag* and possibly, a value for the *flag*, to be placed in the SCCS file. Several **f** options may be supplied on a single *admin* command line. The allowable *flag*s and their values are:

b Allows use of the **−b** option on a *get*(1) command to create branch deltas.

c*rel* (ceiling) The highest release, a number less than or equal to 9999, which may be retrieved by a *get*(1) command for editing. If the **c** flag is omitted, a default value of 9999 is assumed.

f*rel* (floor) The lowest release, a number greater than 0 but less than 9999, which may be retrieved by a *get*(1) command for editing. If the **f** flag is omitted, a default value of 1 is assumed.

d*SID* The default delta number (SID) to be used by a *get*(1) command.

i[*str*] Causes the **No id keywords (ge6)** message issued by *get*(1) or *delta*(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*(1)) are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string; however, the string must contain a keyword and no embedded new-lines.

j Allows concurrent *get*(1) commands for editing on the same SID of an SCCS file, facilitating multiple concurrent updates to the same version of the SCCS file.

l*list* (lock) A *list* of releases to which deltas can no longer be made (get **−e** against one of these *lock* releases fails). The *list* has the following syntax:

<list> ::= <range> <list> , <range>
<range> ::= *RELEASE NUMBER* **a**

The character **a** in the *list* specifies *all releases* for the named SCCS file.

n (null) Causes *delta*(1) to create a *null* delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as anchor points so that branch deltas may later be created from them. If null deltas are not created for skipped releases (ie. **n** is omitted) branch deltas cannot be created from them in the future.

q*text* User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by *get*(1).

m*mod* *Mod*ule name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by *get*(1). If the **m** flag is not specified, *mod* is the name of the SCCS file with the leading **s.** removed.

t*type* *Type* of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by *get*(1).

v[*pgm*]

Causes *delta*(1) to prompt for Modification Request (*MR*) numbers as the reason for creating a delta. The optional *pgm* specifies the name of an *MR* number validity checking program (see *delta*(1)). (If **v** is set when creating an SCCS file, the **m** option must also be used, even if the **m** value is null).

−d*flag* Deletes the specified *flag* from an SCCS file. The **−d** option may be specified only when processing existing SCCS files. Several **−d** options may be supplied on a single *admin* command. See the **−f** option for allowable *flag* names.

l*list* A *list* of releases to be unlocked. See the **−f** option for a description of the **l** flag and the syntax of a *list*.

−a*login* A *login* name, or numerical group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. As many *login*s, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If *login* or group ID is preceded by a ! they are to be denied permission to make deltas.

Several **a** options may be used on a single *admin* command line.

−e*login* A *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several **e** options may be used on a single *admin* command line.

−y[*comment*] The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(1). Omission of the **−y** option results in a default comment line being inserted in the form:

date and time created *YY/MM/DD HH:MM:SS* by *login*

The **−y** option is valid only if the **−i** and/or **−n**

options are specified, i.e., a new SCCS file is being created.

−m[*mrlist*] The list of Modification Requests (*MR*) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*(1). The v flag must be set and if the v flag has a value (the name of an *MR* number validation program), the *MR* numbers are validated. Diagnostics are produced if the v flag is not set or *MR* validation fails.

−h Checks the structure of the SCCS file (see *sccsfile*(5)), and compares a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

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

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

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

FILES

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

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of *ed*(1). The edited file should *always* be processed by an **admin −h** to check for corruption followed by an **admin −z** to generate a proper check-sum. Another **admin −h** is recommended to ensure the validity of the SCCS file.

Admin also uses a transient lock file (called z.*file-name*) to prevent simultaneous updates to the SCCS file by different users. See *get*(1) for further information.

ADMIN(1)

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(4).
Source Code Control System User Guide in the Support Tools Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

SUPPORT STATUS

Supported.

NAME

ar – archive and library maintainer for portable archives

SYNOPSIS

ar key [posname] afile name ...

DESCRIPTION

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

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

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

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive and is useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

v Verbose. Under the verbose option, *ar* gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the files. When used with **x**, it precedes each file with a name.

c Create. Normally *ar* creates *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.

l Local. Normally *ar* places its temporary files in the directory /tmp. This option causes them to be placed in the local directory.

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

Release 2.x archive format is supported permitting usage with archive libraries from Release 2.x transparently.

FILES

/tmp/ar* temporary files

SEE ALSO

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

WARNINGS

Release 2.x archives are supported, but will not be in future releases.

If Release 2.x objects are archived in the new archive format, the external symbol names in the object are mapped to the new release naming conventions before they are entered in the archive global symbol table. See *ld*(1) for further information.

RESTRICTIONS

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

SUPPORT STATUS

Supported.

NAME

as, ljas − common assembler

SYNOPSIS

as [−**o** objfile] [−**n**] [−**m**] [<**m4-args**>] [−**R**] [−**V**] filename
ljas [−**o** objfile] [−**n**] [−**m**] [<**m4-args**>] [−**R**] [−**V**] filename

DESCRIPTION

The *as* command assembles the named file. The *ljas* command is a special version of the *as* assembler. *Ljas* produces *long jump* instructions rather than (short) branch instructions.

OPTIONS

The following options may be specified in any order.

−**o** *objfile*
: Put the output of assembly in *objfile*. By default, the output filename is formed by removing the .s suffix, if there is one, from the input filename and appending a .o suffix.

−**n**
: Turn off long/short address optimization. By default, address optimization takes place.

−**m**
: Run the *m4* macro pre-processor on the input to the assembler. The *m4* −**D** and −**U** options may also be specified, along with other *m4* files containing source or definitions.

−**R**
: Remove (unlink) the input file after assembly is completed. This option is off by default.

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

FILES

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

SEE ALSO

ld(1), m4(1), nm(1), strip(1), a.out(4).
The Assembler in the Support Tools Guide.

WARNING

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

RESTRICTIONS

Arithmetic expressions are permitted to have only one forward referenced symbol per expression.

SUPPORT STATUS

Supported.

NAME

asa − interpret ASA carriage control characters

SYNOPSIS

asa [files]

DESCRIPTION

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

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

Lines beginning with other than the above characters are treated as if they began with a blank. The first character of a line is *not* printed, and an appropriate diagnostic appears on standard error. *asa* forces the first line of each input file to start on a new page.

To view the program output sent to a file with the ASA carriage control characters interpreted, use *asa* thus:

```
asa file
```

To send the program output to a printer properly formatted and pagenated, use *asa* as a filter:

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

SEE ALSO

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

SUPPORT STATUS

Supported.

NAME

ascvt − Release 2.x to new release assembler source translator

SYNOPSIS

ascvt [−o output] filename

DESCRIPTION

The *ascvt* command converts the named Release 2.x assembler source to the new release format.

−o *output* Put the output of the conversion in *output*. By default, the converted output is written to standard out.

WARNING

Ascvt does several transformations on symbols due to changes in the way that the C compiler names symbols. For the new release, C no longer prepends an underscore character to C program symbols. To prevent C program symbols from conflicting with symbols in non-C low level support routines, by convention the % character is appended to all symbols that should not be C callable. The C compiler does not allow % in C symbols. *Ascvt* enforces this, stripping leading underscores and adding % to symbols that do not have underscores.

Numbered local symbols, as supported by the Release 2.x assembler, are not supported by the assembler in this release. To handle cases where these symbols are used, *ascvt* creates unique local labels by combining the last symbol name with the local symbol number appended at the end. The characters **L%** are also prepended so that *as* strips these symbols from the external symbol table.

The C compiler and object utilities support extended length identifiers, a feature known as flexnames. This means that symbols may significantly be much longer than the Release 2.x symbols, which were 8 characters long (7 for C symbols). This may cause some code problems due to misspellings.

RESTRICTIONS

Due to the one pass nature of the assembler in this release, arithmetic expressions with more than one forward referenced symbol cause problems when assembled. *Ascvt* converts the code as it is.

There is no mechanism for specifying string data as there is in the Release 2.x assembler (ie pseudo ops *ascii* and *asciz*). Such constructs are converted using the *byte* pseudo-op, which works for most all cases except the case of SCCS version strings. As *byte* must break up each byte in the string, SCCS control utilities no longer recognize the delta strings.

Negation of constants is not supported by this assembler.

SUPPORT STATUS

Supported.

NAME

at, batch − execute commands at a later time

SYNOPSIS

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

batch

DESCRIPTION

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

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. The allow/deny files consist of one user name per line.

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

An optional *date* may be specified as either a month name followed by a day number (and possibly year number preceded by a 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 valid commands include:

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

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

Batch submits a batch job. It is almost equivalent to **at now**, but not quite. For one, it goes into a different queue. For another, **at now** responds with the error message too late.

OPTIONS

−r 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 only remove your own jobs unless you are the superuser.

−l Lists all jobs scheduled for the invoking user.

EXAMPLES

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

This sequence can be used at a terminal:

```
batch
nroff filename >outfile
<control-d> (hold down control and press d)
```

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

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

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

```
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/queuedefs	scheduling information
/usr/spool/cron/atjobs	spool area

SEE ALSO

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

DIAGNOSTICS

Reports various syntax errors and times out of range.

RESTRICTIONS

The years when specified in the date option is valid for the current year through 1999.

SUPPORT STATUS

Supported.

NAME

ati − read and write ANSI format tapes

SYNOPSIS

ati −i [eftv] [−F devname**] [**patterns**]**

ati −o [exv] [−n [name**]] [−F** devname**] [−d** fset**] [−b** size**] [−r** size**] [−i** volid**] [−c** fsec**] [−s** lnum**] [−q** fnum**] [ebcdic | ibm]**

DESCRIPTION

Ati reads and writes ANSI format tapes installed in a Small Computer Systems Interface (SCSI) tape drive.

Ati **−i** (in) copies the files specified by *patterns* from the tape specified by *devname* to the current directory. *Ati* provides no directory support and all files are copied into the current directory. File name patterns follow the name-generating notation of *sh*(1). If no file names are given, all files are copied from the tape; that is, the default *pattern* is *. *Ati* determines if the tape is encoded in EBCDIC or ASCII and uses the appropriate character set. If *ati* finds an existing file in the current directory with the same name, the user is warned of the dilemma and has 15 seconds to terminate *ati*. If *ati* is not terminated, the file in the current directory is overwritten by the file on the tape. If the first file on the tape has a file section number greater than 1, *ati* appends the tape file data to the end of the existing file if one exsists or issues a warning if the file does not exist.

Ati **−o** (out) reads file names from the standard input and copies those files to the tape specified by *devname*.

OPTIONS

b *size*
: The maximum block size to use on the tape (**−o** only). The *size* may be followed by **k**, **b**, or **w** to speicfy multiplication by 1024, 512, or 2, respectively. The default *size* is 512 bytes; the maximum is 64K bytes.

c *fsec*
: The first file on the tape is a continuation (**−o** only). The *fsec* is the file section number in the HDR1 field for the first file on the tape. *Ati* does not check the validity of *fsec*. This option is only needed if the user runs out of blank tapes and has to terminate the copy and continue it at a later time.

d *fset*
: The file set identification for the copied files (**−o** only). The default *fset* is the volume identification.

e
: Stop the copy operation at end of volume. The default is to stop the copy operation at the end of the file set.

f
: Negate the patterns; copy in all files with names that do not match any pattern (**−i** only).

F *devname*
: The tape device to use. The default device name is **/dev/rmt/0myy**.

i
: The volume identification for the tape (**−o** only). The default volume identification is 000000.

n *name*
The owner identification of a new tape (**—o** only). If **n** is not entered, *ati* skips to the logical end of data on the tape before adding new files. If **n** is entered, *ati* overwrites any existing data on the tape. The *name* cannot start with a hyphen character (-).

q *fnum*
The file sequence number of the first tape in the volume (**—o** only). Use with the **c** and **s** options for multi-volume tapes.

r *size*
The record size to use on the tape (**—o** only). The *size* may be followed by **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2, respectively.

- For fixed length records, this is the actual record size. The default is a record size equal to the block size. The record size must be an equal divisor of the block size. Records shorter than this size are padded with space characters; records longer than this size are split to fit the specified record size.
- For variable length records, this is the maximum record size not including the 4-byte variable length indicator. The default and the maximum is 4 less than the block size. Records shorter than this size are not padded. Records longer than this size are split to fit the specified record size.

s *lnum*
Skip to the *lnum*th character in the first file before writing on the tape (**—o** only). This option is used with the **c** option to split a file across two or more tapes. *Ati* does not check the validity of *lnum*. The user must take care to not create invalid tapes.

t Print a directory of the files on the tape (**—i** only). No copy operation occurs. If **v** is also entered, print the tape volume label and file information for each file on the tape.

v Print the names of all files to standard output as they are copied if a copy is occurring or print the volume label and file information if a directory print (see **t** option) is occurring.

x The tape contains fixed length records (**—o** only). The default is variable length records with a 4-byte decimal variable length indicator at the beginning of each record.

ebcdic
Write the tape in EBCDIC (**—o** only). The default is ASCII.

ibm Write the tape in IBM EBCDIC. This mapping corresponds better to some IBM train printers.

FILES
/dev/rmt/*

EXAMPLES
To copy all files on the tape in /dev/rmt/0myy to the current directory, enter

```
ati —i
```

To copy only *file1* and *file2* from the tape in /dev/rmt/0myy to the current directory, enter

```
ati −i file1 file2
```

To enter a list of file names from standard input of files to be copied to the tape in /dev/rmt/0myy, enter

```
ati −o
```

To copy a set of files using the list of file names in *filelist* to a tape in /dev/rmt/0myy with volume identification of 000100 in EBCDIC with 80 byte variable length records in 4K byte blocks, enter

```
ati −ov −n PAY1 −b 4k −r 80 −i 000100 −ebcdic < filelist
```

SEE ALSO

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

SUPPORT STATUS

Supported.

NAME

awk − pattern scanning and processing language

SYNOPSIS

awk [−Fc] [prog] [parameters] [files]

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that is 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

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 double 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)).

Built-in functions *exp*, *log*, and *sqrt* are available, as well as the following functions:

length returns the length of its argument taken as a string. If no argument is given, *length* returns the length of the whole line.

int returns its argument truncated to an integer.

substr (*s* , *m* , *n*)
returns the *n*-character substring of *s* that begins at position *m*.

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), lex(1), sed(1), malloc(3X).
The awk Programming Language in the Support Tools Guide.

RESTRICTIONS

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.

SUPPORT STATUS

Supported.

NAME

backup, restore - backup or restore selected files

SYNOPSIS

backup
restore

DESCRIPTION

The *backup* and *restore* commands permit a user to backup and restore selected files to and from floppy disk, streaming tape, or magnetic tape through a menu interface.

The *backup* command provides a menu interface for the selection of files to be written to the backup media, the backup media to be used, and the type of backup desired. A backup can be either full, where all files in the requested directory are copied, or incremental, where only files modified in a specified number of days (1-100) are copied to the backup media. The backup of an individual file is assumed to be a full backup.

The *restore* command also provides a menu interface for the selection of files to restore from the backup media, the media to be used, and the type of restore required. A restore can be either unconditional, where all specified files are copied from the backup media, or conditional, where the specified files are copied from the backup media only if the backup version is newer than the system version.

RESTRICTIONS

The *backup* and *restore* commands are done via *cpio*(1) and the restrictions and limitations of *cpio* are also present in these utilities.

Backups may be performed in multi-user mode, but care should be taken to insure that the files intended for backup are not being modified.

SEE ALSO

cpio(1).

SUPPORT STATUS

Supported.

NAME

banner − make posters

SYNOPSIS

banner strings

DESCRIPTION

Banner prints its arguments, one per line, in large letters on the standard output. Each argument may be a maximum of 10 characters long. Enclose any argument which contains a space character or special character in double quotation marks.

EXAMPLES

To display GOOD MORNING on the terminal enter:

```
banner "good morning"
```

To print GOOD MORNING on the printer enter:

```
banner "good morning" | print -b
```

SEE ALSO

echo(1).

SUPPORT STATUS

Supported.

NAME

basename, dirname — deliver portions of path names

SYNOPSIS

basename string [suffix]
dirname string

DESCRIPTION

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

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

EXAMPLES

The following example, invoked with the argument **/usr/src/cmd/pgm.c**, compiles the named file and moves the output to a file named **pgm** in the current directory:

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

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

```
NAME=`dirname /usr/src/cmd/pgm.c`
```

SEE ALSO

sh(1).

RESTRICTIONS

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

SUPPORT STATUS

Supported.

NAME

bc − arbitrary-precision arithmetic language

SYNOPSIS

bc [−c] [−l] [file ...]

DESCRIPTION

bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. *bc* inputs any specified *files* first, then it reads standard input. *bc* prints the value of input statements that are expressions, except when the main operator is an assignment. *bc* is actually a preproccesor for *dc*(1), which it invokes automatically, unless option **−c** is used.

OPTIONS

−c (compile only) sends output from *bc* to standard output, rather than to *dc*. If **−c** is not present, *dc* is invoked automatically.

−l includes the arbitrary precision math library in the input files. 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

SYNTAX

In the following syntax descriptions, **L** means any letter a−z, **E** means expression, and **S** means statement.

Comments

/* comment text */

Names

simple variables: **L**
array elements: **L [E]**
ibase
obase
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) call to user-defined function

Operators

+ − * /	arithmentic operators
%	remainder
^	power
++ −−	prefix and postfix increment and decrement; apply to names

```
< > == <= >= !=     comparison operators
= =+ =- =* =/ =% =^
```

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 )
}
```

SEMANTICS

All function arguments are passed by value.

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. Variables denoted *auto* in functions are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, follow the array name with empty square brackets.

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).

Arbitrary Precision Desk Calculator Language (BC) in the Support Tools Guide.

RESTRICTIONS

No &&, || yet.

For statement must have all three Es.

Quit is interpreted when read, not when executed.

SUPPORT STATUS

Supported.

NAME

bdiff − big diff

SYNOPSIS

bdiff file1 file2 [n] [−s]

DESCRIPTION

Bdiff locates the lines in two files which must be changed to bring the two files into agreement. The purpose of *bdiff* is to process files which are too big for *diff*. *Bdiff* ignores lines common to the beginning of both files, splits the remainder of each file into *n*−line segments, and invokes *diff* upon corresponding segments in the two files.

If *file1* or *file2* is −, *bdiff* reads the standard input for that file.

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.

OPTIONS

n This numeric option denotes the maximum number of lines in each segment. The default value, 3500 lines, is adequate, except when the 3500-line segments are too large for *diff* (causing *bdiff* to fail).

−s This *silence* option suppresses diagnostics printed by *bdiff*. Diagnostics from *diff* are not suppressed, however.

Note: *n* must precede −s if both options are used.

FILES

/tmp/bd?????

SEE ALSO

diff(1).

DIAGNOSTICS

Use *help*(1) for explanations.

SUPPORT STATUS

Supported.

BFS(1)

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 (the maximum possible size) 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* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional − suppresses printing of sizes.

Input is prompted with * if **P** and a carriage return are typed as in *ed*. Prompting can be turned off again by inputting another **P** and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols besides / and ?: > indicates downward search without wrap-around, and < indicates upward search without wrap-around. There is a slight difference in mark names: only the letters **a** through **z** may be used, and all 26 marks are remembered.

COMMANDS

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

xf *file*
: Indicates that future commands are read from the named *file* until an end-of-file is reached, an interrupt signal is received or an error occurs. Reading then resumes with the file containing the **xf** command. 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*]
: Diverts further output from the **p** and null commands to *file*, which, if necessary, is created mode 666. If *file* is not specified, **xo** diverts the output to standard output. Note that each diversion causes truncation or creation of the file.

: *label*
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.

(*addr1* , *addr2*)**xb**/*regular expression*/*label*
Branches to *label* if the command succeeds. *Xb* fails under any of the following conditions:

- Either address is not between 1 and $.
- The second address is less than the first.
- The regular expression does not match at least one line in the specified range, including the first and last lines.

When successful, *xb* sets *dot* to the line matched and branches to *label*. *Xb* is the only command 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 branch. The **xb** command is an invalid command when read from a terminal. If it is read from a pipe only a forward branch is possible.

xt *n* Truncates output from the **p** and null commands to at most *n* characters. Initially *n* is set to 255.

xv[*varname*][*spaces*][*value*]
Assigns *value* to *varname*. *Varname* is a single digit (0-9). If ! precedes *value*, *value* is intrepreted as a UNIX system command, and the output generated from the execution of that command is stored in *varname*.

To reference a variable, put a % in front of the variable name. To escape the special meaning of %, or ! a \ must precede it.

Example 1

```
xv4100
```

assigns 100 to the variable 4.

```
xv5 100
```

assigns 100 to the variable 5.

```
Xv61,100p
```

assigns the value **1,100p** to the variable **6**.

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

all print the first 100 lines.

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

globally searches for the characters **100** and prints each line containing a match.

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

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

Example 2

```
.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.

Example 3

```
xv7\!date
```

stores the value **!date** into variable **7**.

xbz *label*

xbn *label*

Tests 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**.

Example 1

```
xv55
: l
/size/
xv5!expr %5 − 1
!if 0%5 != 0 exit 2
xbn l
```

Example 2

```
xv45
: l
/size/
xv4!expr %4 − 1
!if 0%4 = 0 exit 2
```

BFS(1)

xbz l

xc [*switch*]

Compresses output from the **p** and null commands if *switch* is 1; does not compress if *switch* is 0. Without an argument, **xc** reverses the current value of *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

SEE ALSO

csplit(1), ed(1), regcmp(3X).
Bif File Scanner (bfs) in the Editing Guide.

DIAGNOSTICS

Self-explanatory error messages are printed when prompting is on. ? replaces error messages if prompting is turned off.

SUPPORT STATUS

Supported.

NAME

bs — a compiler/interpreter for modest-sized programs

SYNOPSIS

bs [file [args]]

DESCRIPTION

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

Bs normally receives its input from the console. However, if the *file* argument is specified when the compiler/interpreter is invoked, *bs* first reads statements from *file*, compiling them for later execution. *Bs* then reads statements from the console, executing them immediately. The result of each immediate expression statement is printed, unless the final operation is an assignment.

Bs statements are terminated by new-line. A backslash escapes the new-line, however, allowing a *bs* statement to continue to the next line. *Bs* statements have two forms:

statement
label statement

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

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

STATEMENT SYNTAX

expression
Expression syntax follows the description of statement syntax.

break
Exits from the inner-most *for/while* loop.

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

compile [file_expr]
Signals that succeeding statements are to be compiled, overriding the immediate execution default. The optional *file-expr* is evaluated and used as a file name for further input, and a *clear* is executed immediately afterwards. This command is executed immediately.

continue
Transfers to the loop-continuation of the current *for/while* loop.

dump [varname]
Prints the name and current value of every non-local variable. If *varname* is specified, only *varname* and its value are printed. After an error or interrupt, the number of the last statement and the user-function trace are displayed.

exit [**expression**]
Returns to system level, with *expression* returned as process status.

execute
Puts the console in the immediate execution mode (an interrupt has a similar effect). *Execute* does not cause stored statements to execute (use *run* below).

for
for var_name = init_expr limit_expr statement
for var_name = init_expr limit_expr
...
next
Executes a statement (first form) or a group of statements (second form) under control of a named variable repeatedly. *Var_name* takes on the value of the *init_expr*, then is incremented by one on each loop, until its the value exceeds *limit_expr*.

for init_expr , test_expr , action_expr statement
for init_expr , test_expr , action_expr
...
next
These forms require three expressions separated by commas: *init_expr* is the initialization, *text_expr* must be true to continue looping, and *action_expr* is the loop-continuation action, normally an increment.

fun
fun *fname*([arg, ...]) [var, ...]
...
nuf
Defines a user-written function *fname*, its arguments, and its local variables. Up to a total of ten arguments and/or local variables are allowed. Neither arguments or local variables can be arrays, nor can they be I/O associated. Function definitions may not be nested.

freturn
Signals the failure of a user-written function. If the user-written function was called using the interrogation operator, *freturn* transfers to the expression given in the interrogation statement, possibly by-passing intermediate function returns. If the interrogation operator was not used, *freturn* returns zero.

goto label-name
Passes control to the internally stored statement with the label *label-name*.

ibase *N*
Sets the input base (radix) to *N*. The only supported values for *N* are 8, 10 (the default), and 16. Hexadecimal values

10−15 are entered as **a**−**f**. The first character in a hexadecimal number must be a digit (i.e., **f0a** must be entered as **0f0a**). *Ibase* is executed immediately.

if

if *expression* statement
if *expression*
...
[**else**
...]
fi
if *expression*
...
elif *expression*
...
[elif *expression*]
...
fi

Executes *statement* (first form) or group of statements (second and third forms) if *expression* evaluates to non-zero. The *0* string and the *null* string evaluate as zero. An optional *else* (second form), or *else if* (third form), allows for a group of statements to be executed when the first group is not. The only statement permitted on the same line with an *else* is an *if*; only other *fi*'s can be on the same line with a *fi*. The third form is a shorthand for nested *if* statements.

include file_expr

Includes the *bs* source statements in the file *file_expr* in the program being compiled. *File_expr* must evaluate to a filename, and that file must contain *bs* source statements. *Include* statements may not be nested.

obase *N*

Sets the output base (radix) to *N* (see *ibase* above). *Obase* is executed immediately.

onintr label
onintr

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

return [expression]

Evaluates *expression* and passes the result back as the value of a function call. If *expression* is not given, zero is returned.

run Resets the random number generator, and passes control to the first internal statement. If the *run* statement is contained in a file, it should be the last statement.

stop Stops execution of internal statements and reverts *bs* to immediate mode.

trace [expression]

Turns off function tracing if the expression is null or evaluates to zero. Otherwise, prints a record of user-function calls/returns. Each *return* decrements the *trace* expression value.

while

while test-expr statement
while test-expr
...
next

Executes the statement (first form) or a group of statements (second form) until *test_expr* evaluates to zero or nullstring.

! shell command

Escapes to a shell command, after which execution of *bs* continues.

... Denotes a comment line.

EXPRESSION SYNTAX

label_name
var_name

Specifies a variable or label. Both are composed of a letter (upper or lower case) followed by zero or more letters and digits. Only the first six characters of a name are significant. All names are global to the program, except for names declared in *fun* statements. Variables can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function *open*( ) below).

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

Invokes the user-written or built-in function with *fname*. If *fname* is not a built-in function (listed below), *fname* must be defined with a *fun* statement. Arguments to functions are passed by value.

var_name [expression [, expression] ...]

References either an array or a table (see built-in *table* functions below). If *var_name* denotes an array, each expression is truncated to an integer and used as an array index. An array reference is syntactically identical to a name; a[1,2] is the same as a[1][2]. The array indices must be between 0 and 32767.

number

Represents a constant value. A number contains digits, an optional decimal point, and possibly a scale factor consisting of an e followed by a possibly signed exponent.

string

Character strings are delimited by double quotes. The backslash escapes the double quote (\"), newline (\n), carriage return (\r), backspace (\b), and tab (\t), allowing these characters to appear in a string. Otherwise, backslash stands for itself.

(expr)

Parentheses are used to alter the normal order of evaluation.

($expr^0$, $expr^1$[, $expr^2$ ...]) [subscript-expr]

subscript_expr is used as a subscript to select an expression *expr* from the parenthesized list. List elements are numbered from the left, starting at zero. The expression:

(False, True)[a == b]

has the value True if the comparison is true. The statement:

goto (read_lbl,write_lbl,exit_lbl) [menu_selection-1]

transfers control to the statement marked *read_lbl* if menu-selection is 1, to statement *write_lbl* if menu_selection is 2, and to statement *exit_lbl* if menu_selection is 3.

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

− expression
Returns the negation of the expression.

++ name
Returns the incremented value of the variable (or array reference).

−− name
Returns the decremented value of the variable.

! expression
Returns the logical negation of the expression. Watch out for the shell escape command. (See ! in statement syntax.)

expression *operator* expression
Invokes a binary function symbolized by *operator* on the two expressions. Both operands are converted to numeric form before the operator is applied, except for the assignment, concatenation, and relational operators.

BINARY OPERATORS

The binary operators are listed in increasing precedence. Assignment binds right to left, all other operators bind left to right.

= The assignment operator. Binds the left operand (a name or an array element) to the right operand. The result is the right operand.

Underscore. The concatenation operator.

& Logical *and* operator. Returns zero if either of its arguments are zero; returns one if both of its arguments are non-zero. & treats a null string as a zero.

| Logical *or* operator. Returns zero if both of its arguments are zero; returns one if either of its arguments is non-zero. | treats a null string as a zero.

< <= > >= == !=
The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, != not equal to). Return one if the operands are related as specified by the operator, and return zero otherwise. Relational operators at the same level extend as follows: *a>b>c* is the same as *a>b & b>c*. A string comparison is made if both operands are strings.

+ −
Addition and subtraction operators.

* / %
Multiplication, division, and remainder operators.

^ Exponentiation operator.

BUILT-IN FUNCTIONS

Argument Reference Functions

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

narg()
Returns the number of arguments passed on the current function call level. At level zero *narg* returns the command argument count.

Mathematical Functions

abs(x)
Returns the absolute value of x.

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

ceil(x)
Returns the smallest integer not less than x.

cos(x)
Returns the cosine of x in radians.

exp(x)
Returns the exponential function of x.

floor(x)
Returns the largest integer not greater than x.

log(x)
Returns the natural logarithm of x.

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

sin(x)
Returns the sine of x in radians.

sqrt(x)
Returns the square root of x.

String Operations

size(string)
Returns the size (length in bytes) of *string*.

format(fmt, name)
Returns the formatted value of *name*. *Fmt* is assumed to be a format specification: %...f, %...e, or %...s, as described in *printf*(3S).

index(string1, string2)
Returns the number of the first position in *string1* that any

of the characters from *string2* matches. No match yields zero.

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

substr(string, start, length)
Returns the sub-string of *string* defined by the starting position *start* and *length*.

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

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

File Handling

open(fname, file, mode)
Associates *fname* (a *bs* variable name passed as a string) with a file opened for a particular mode of output. *File* must be one of the following:

0	standard input
1	standard output
2	error output
string	filename
!*string*	command to be executed (via *sh* −*c*)

Mode must be one of the following:

r	read
w	write
W	write without new-line
a	append

The initial associations are:

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

Examples are given in **Programming Tips.**

close(name)
Removes the association betwen *fname* (a variable name passed as a string) and the file connected to it by a previous *open* statement.

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

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

Tables

table(tname, tsize)
Associates *tname* with a table of size *tsize*, where *tname* is a *bs* variable name passed as a string, and *tsize* is the minimum number of elements to be allocated. The table allocated is an associatively accessed single-dimension array whose subscripts or *keys* are strings (numbers are converted to strings). *Bs* prints an error message and stops on table overflow.

item ***(tname,i)***
key()
Item returns the value of table elements accessed sequentially (in normal use, there is no orderly progression of key values). *Key* returns the subscript or *key* of the previous *item* call. The *tname* argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table, for example:

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

iskey ***(tname,keyword)***
Returns one if the key **keyword** exists in the table **tname**, and zero otherwise.

Miscellaneous

eval(string)
Evaluates *string* as a *bs* expression. *Eval* is handy for converting numeric strings to numeric internal form. *Eval* can also be used as a crude form of indirection, as in:

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

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

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

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

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

plot(request, args)

Produces output on devices recognized by *tplot*(1G). The following are valid requests:

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

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

last()
Returns the most recently computed value if *bs* is in immediate mode.

PROGRAMMING TIPS

Using *bs* as a calculator:

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

#    Compound interest (6% for 5 years on $1,000).
int = .06 / 4
bal = 1000
for i = 1 5*4  bal = bal + bal*int
bal - 1000
346.855007
...
exit
```

The outline of a typical *bs* program:

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

Input/Output examples:

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

#    Pipe between commands.
open("ls", "!ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls) ...
```

```
...
# be sure to close (wait for) these:
close("ls")
close("pr")
```

SEE ALSO

ed(1), sh(1), tplot(1G), access(2), printf(3S), stdio(3S), plot(4).

See Section 3 of the *Programmer Reference Manual* for further description of the mathematical functions (*pow* on *exp*(3M) is used for exponentiation); *bs* uses the Standard Input/Output package.

SUPPORT STATUS

Plot functions are not supported. All other functions are supported.

NAME

cal – print calendar

SYNOPSIS

cal [[month] year]

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. *Year* must be between 1 and 9999. The *month* is a number between 1 and 12.

EXAMPLE

To print a calendar for 1986 enter:

```
cal 1986 | print -b
```

RESTRICTIONS

The first month of the calendar year is assumed to be January for all years, even though that assumption is historically inaccurate.

Note that **cal 86** does not refer to the year 1986; it refers to the year 86.

SUPPORT STATUS

Supported.

NAME

calendar − reminder service

SYNOPSIS

calendar [−]

DESCRIPTION

Calendar consults the file **calendar** in the current directory and prints out lines that contain *today*'s or *tomorrow*'s date anywhere in the line. On weekends *tomorrow* includes all days through Monday. Most month-day dates are recognized, but day-month dates are not. Recognized formats include:

12/7 Dec. 7 december 7

When the − option is present, *calendar* checks the calendar of every user who has a file **calendar** in their login directory and sends them any positive results by *mail*(1). Normally *calendar* is run daily by *cron*(1).

EXAMPLE

To receive mail to remind you of your appointments and meetings, create a file named **calendar** in your login directory. Edit the file periodically to add and remove lines such as:

6/8 Atlas project meeting 2pm, Annex.
6-20 Carter lunch 12:30, Surf Inn.

FILES

calendar
/usr/lib/calprog to determine dates for *today* and *tomorrow*
/etc/passwd
/tmp/cal*

SEE ALSO

crontab(1), mail(1).

RESTRICTIONS

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

SUPPORT STATUS

Supported.

NAME

cancel – cancel requests to an LP line printer

SYNOPSIS

cancel [ids] [printers]

DESCRIPTION

Cancel cancels line printer requests that were made by the *lp*(1) command. The command line arguments may be either request *ids* (as returned by *lp*(1)) or *printer* names (for a complete list, use *lpstat*(1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

FILES

/usr/spool/lp/*

SEE ALSO

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

SUPPORT STATUS

Not supported.

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. If no input file is given, or if the argument − is given, *cat* reads from the standard input file. No input file may be the same as the output file unless it is a special file.

OPTIONS

−u Do not buffer output. (Output is normally buffered.)

−s Silence messages about nonexistent files.

−v Visibly print non-printing characters, with the exception of tabs, new-lines and formfeeds:

- Print control characters as **^X** (control-x).
- Print the DEL character (octal 0177) as **^?**.
- Print non-ASCII characters (with the high bit set) as M-*x* where *x* is the character specifed by the seven low order bits.

−t Print tab characters as **^I** (effective only when used with the −v option).

−e Print a $ character at the end of each line preceding a newline (effective only when used with the −v option).

WARNING

Use shell special characters carefully because command formats such as

```
cat file1 file2 > file1
```

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

EXAMPLES

To display a file, enter

```
cat file
```

To concatenate *file1* and *file2* and place the result in *file3* enter

```
cat file1 file2 > file3
```

SEE ALSO

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

SUPPORT STATUS

Supported.

NAME

cb − C program beautifier

SYNOPSIS

cb [−s] [−j] [−l leng **] [** file ... **]**

DESCRIPTION

cb reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code. If no options are specified, *cb* preserves all user new-lines.

OPTIONS

−s	canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*
−j	rejoins split lines
−l	leng splits lines that are longer than *leng* .

SEE ALSO

cc(1).

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

C Language in the Programming Guide.

RESTRICTIONS

Punctuation that is hidden in preprocessor statements causes indentation errors.

SUPPORT STATUS

Supported.

NAME

cc, cc.10 − C compiler

SYNOPSIS

cc [option] ... file ...
cc.10 [option] ... file ...

DESCRIPTION

Cc or *cc.10* is the UNIX system C compiler.

16-Bit Systems

On 16-bit systems, *cc* is the C compiler which generates object to run on the 16-bit system (68010 processor). There is no compiler to generate object to run on a 32-bit system.

32-Bit Systems

On 32-bit systems, *cc* is the C compiler which generates object to run on the 32-bit system (68020 processor); *cc.10* is the C compiler which generates object to run on the 16-bit system (68010 processor).

The default *cc* command produces object which includes 68020 instructions. These objects cannot be executed on a 16-bit system.

The *cc.10* command produces object which includes only 68010 instructions. These objects can be executed on a 16-bit system or a 32-bit system. The *cc.10* command can be executed by entering *cc.10* as the command or by changing your path

```
PATH=:/bin.10$PATH
export PATH
```

and entering *cc* as the command.

FILE ARGUMENTS

Cc accepts several types of file arguments. Files whose names end with .c are taken to be C source programs. They are compiled, and each object program is left in the file whose name is that of the source with .o substituted for .c. The .o file is normally deleted, however, if a single C program is compiled and loaded in the same command.

Files whose names end with .s are taken to be assembly source programs and are assembled, producing a .o file.

OPTIONS

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

−c Suppress the link edit phase of the compilation and force an object file to be produced even if only one program is compiled.

−p Produce code that counts the number of times each routine is called; also, if link editing takes place, replace the standard startoff routine by one that automatically calls *monitor*(3C) at the start and writes out a **mon.out** file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1).

−g Cause the compiler to generate additional information needed for the use of *sdb*(1). *Nm*(1) and *dump*(1) also display this

Floating Point Handling

The 16-bit system Release 2.x floating point support option (see −f above) is primarily for use with existing Release 2.x application packages that have floating point code in them. This defines the symbol **_IEEE** with a value of 0 to cause the preprocessor to select Release 2.x floating point values, cause the compiler to generate floating point constants in Release 2.x format, and link with the Release 2.x floating point version of the C library. The default defines **_IEEE** to a value of 1, generates **IEEE** constants, and links with the **IEEE** support libraries.

This option should be used when compiling any module to be linked with 16-bit system Release 2.x applications, objects, or libraries. See *flcvt*(3) for routines to convert between Release 2.x and this release floating point formats.

DIAGNOSTICS

The diagnostics produced by C itself are self-explanatory. Occasional messages may be produced by the assembler or the link editor.

SUPPORT STATUS

Supported.

NAME

cd − change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

Cd changes the working or current directory to the specified directory. *Cd* must have execute (search) permission in *directory*.

If *directory* is not specified, the value of shell parameter **$HOME** (see *sh*(1)) is used as the new working directory. Normally the value in **$HOME** is the name of the home or login directory.

If *directory* specifies a complete pathname starting with */*, . (the current directory), or .. (the parent directory of the current directory), *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 **$CDPATH** (see *sh*(1)) shell variable. **$CDPATH** has the same syntax as, and similar semantics to, the **$PATH** shell variable.

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 by and is internal to the shell.

EXAMPLES

To change the working directory to the parent directory of the current working directory, enter

```
cd ..
```

To change the working directory to your login directory, enter

```
cd
```

SEE ALSO

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

SUPPORT STATUS

Supported.

NAME

cdc − change the delta commentary of an SCCS delta

SYNOPSIS

cdc −rSID [−m[mrlist]] [−y[comment]] files

DESCRIPTION

Cdc changes the delta commentary for the delta specified by the −r*SID* argument of each named SCCS file.

Delta commentary is the Modification Request (MR) and comment information normally specified via the *delta*(1) command using the −m and −y option.

If a named file is a directory, *cdc* processes each file in the directory as if it were a named file, silently ignoring unreadable files and non-SCCS files (files whose final suffix does not begin with s).

If the file name − is given, *cdc* reads standard input (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

MR numbers can be added and deleted using the −m option. If −m is not used, and standard input is a terminal, *cdc* prompts for MR changes with MRs? on the standard output before reading standard input. If standard input is not a terminal, no prompt is issued.

After the prompt, enter a list of MRs separated by blanks or tabs, and terminated with an unescaped new-line character. A null list has no effect.

MRs are added to the list of MRs in the same manner as *delta*(1).

To delete an MR, precede the MR number with an !. (see *EXAMPLES*). If the MR to be deleted is currently in the list of MRs, *cdc* removes it and changes it into a comment line. *Cdc* places a list of all deleted MRs in the comment section and precedes the list with a comment line indicating they were deleted.

Note that if the v flags of the SCCS file has a value, *cdc* takes that value to be the name of a program which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, *cdc* terminates and the delta commentary remains unchanged.

Comment information can be replaced using the −y option. If −y is not used, and standard input is a terminal, *cdc* prompts comments? on the standard output before reading standard input. If the standard input is not a terminal, no prompt is issued.

Comment(s) entered at this prompt replace the existing comment(s) for the delta specified by −r*SID*. The existing comments are kept and preceded by a comment line stating that they were changed. A null comment has no effect.

The exact permissions necessary to modify the SCCS file are documented in the *Source Code Control System User Guide*, but essentially, if you made the delta, or if you own the file and directory, you are permitted to modify the commentary.

OPTIONS

Arguments to *cdc*, both options and filenames, may appear in any order. All options described below apply independently to each named file:

−r*SID* Specifies the SCCS IDentification string (SID) of a delta for which the delta commentary is to be changed.

−m*mrlist* Adds and/or deletes *mrlist*, a list of MR numbers, in the delta commentary of the SID specified by −r*SID*. The numbers in the list are separated by blanks; the entire list is enclosed in double quotes.

−y*comment*
Replaces the comment(s) already existing for the delta specified by −r*SID* with *comment*, arbitrary text terminated by an unescaped new-line.

EXAMPLES

```
cdc −r1.6 −m"b178-12345 !b177-54321 b179-00001"
−ytrouble s.file
```

adds b178-12345 and b179-00001 to the MR list, removes b177-54321 from the MR list, and adds the comment **trouble** to delta 1.6 of s.file.

```
cdc −r1.6 s.file
MRs? !b177-54321 b178-12345 b179-00001
comments? trouble
```

does the same thing.

WARNINGS

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

FILES

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

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).
Source Code Control System User Guide in the Support Tools Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

SUPPORT STATUS

Supported.

NAME

cflow − generate C flow graph

SYNOPSIS

cflow [−r] [−ix] [−i_] [−dnum] files

DESCRIPTION

Cflow analyzes a collection of C, YACC, LEX, assembler, and object files and attempts to build a graph charting the external references. Files suffixed in .y (for YACC), .l (for LEX), .c (for C), and .i are preprocessed (bypassed for .i files) as appropriate and then run through the first pass of *lint*(1). The −I, −D, and −U options of the C-preprocessor are recognized. Files suffixed with .s are assembled and information is extracted (as in .o files) from the symbol table. The output of all this non-trivial processing is collected and turned into a graph of external references which is displayed upon the standard output.

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

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

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

```
int     i;

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

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

the command

```
cflow file.c
```

produces the the output

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

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

OPTIONS

The following options are interpreted by *cflow*:

−r Reverse the "caller:callee" relationship to produce an inverted listing showing the callers of each function. The listing is sorted in lexicographical order by callee.

−ix Include external and static data symbols. The default is to include only functions in the flow graph.

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

−dnum
Terminate the flow graph at the level specified by the *num* decimal integer. By default this is a very large number. The cutoff depth should not be set to a nonpositive integer.

DIAGNOSTICS

Error messages warn about bad options. Multiple definitions produce an error message, and only the first definition is accepted. Other messages may come from the various programs used (e.g., the C-preprocessor).

SEE ALSO

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

RESTRICTIONS

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

SUPPORT STATUS

Supported.

NAME

chown, chgrp − change owner or group

SYNOPSIS

chown owner file ...
chgrp group file ...

DESCRIPTION

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

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

If either *chown* or *chgrp* is invoked by other than the superuser, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, are cleared.

EXAMPLE

To change the group of all files in */usr/acct/thompson/project* to *thor*, enter

```
chgrp thor /usr/acct/thompson/project/*
```

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1), chown(2), group(4), passwd(4).

SUPPORT STATUS

Supported.

NAME

clear – clear terminal screen

SYNOPSIS

clear

DESCRIPTION

Clear clears your screen if possible.

Clear looks in the environment for the terminal type and then in */etc/termcap* to determine how to clear the screen.

FILES

/etc/termcap terminal capability data base

SUPPORT STATUS

Supported.

NAME

cmp − compare two files

SYNOPSIS

cmp [−l] [−s] file1 file2

DESCRIPTION

Cmp compares the two files. If *file1* is −, *cmp* uses the standard input. Normally, *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 (i.e. *file1* consists entirely of the first twenty lines of *file2*), *cmp* notes that fact and prints an end-of-file message.

OPTIONS

−l Prints the byte number (decimal) and the differing bytes (octal) for each difference.

−s Prints nothing for differing files; return codes only.

SEE ALSO

comm(1), diff(1).

EXIT CODES

0 identical files
1 different files
2 inaccessible or missing file argument

SUPPORT STATUS

Supported.

NAME

col − filter reverse line-feeds

SYNOPSIS

col [−bfpx]

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.

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.

Col converts white space to tabs on output wherever possible to shorten printing time.

Col assumes the ASCII control characters SO (\017) and SI (\016) to start and end text in an alternate character set. *Col* remembers the character set to which each input character belongs, and generates on output SI and SO characters 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 I/O filter programs. *Col* ignores all other non-printing characters.

Normally, *col* ignores any unknown escape sequences found in its input.

OPTIONS

−b Assumes that the output device in use is not capable of backspacing. If two or more characters are to appear in the same place, only the last one is output.

−f Indicates output from *col* may contain forward half-line-feeds (ESC-9), but will still never contain either kind of reverse line motion.

−x Suppresses conversion of white space to tabs

−p Outputs unknown escape sequences found in its input 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 sequence.

SEE ALSO

nroff(1), tbl(1).

NOTES

The input format accepted by *col* matches the output produced by *nroff* with either the **−T37** or **−Tlp** options. Use **−T37** (and the **−f** option of *col*) if the ultimate disposition of the output of *col*

will be a device that can interpret half-line motions, and **-Tlp** otherwise.

RESTRICTIONS

Col cannot back up more than 128 lines.
A line may contain at most 800 characters, including backspaces.
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.

SUPPORT STATUS

Supported.

NAME

comb − combine SCCS deltas

SYNOPSIS

comb [−o] [−s] [−psid] [−clist] files

DESCRIPTION

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

The generated shell procedure is written on the standard output.

OPTIONS

Each option is explained as though only one named file is to be processed, but the effects of any option apply independently to each named file.

−p*sid* Discards all deltas in the reconstructed file that are older than the delta with SCCS Identification string *SID*.

−c*list* Preserves deltas in *list* (see *get*(1) for the syntax of a *list*) and discards all others.

−o For each get −e generated, accesses the reconstructed file at the release of the delta to be created, otherwise the reconstructed file is accessed at the most recent ancestor. Use of the −o option may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

−s Generates a shell procedure which, when run, produces a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 * (original − combined) / original

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

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

FILES

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

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).
Source Code Control System User Guide in the Support Tools Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

RESTRICTIONS

Comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

SUPPORT STATUS

Supported.

NAME

comm − select or reject lines common to two sorted files

SYNOPSIS

comm [− [123]] file1 file2

DESCRIPTION

Comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort*(1)). *Comm* 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. (*File1* or *file2* may be −, but not both.)

Flags 1, 2, or 3 suppress printing of the corresponding columns. Thus **comm −12** *file1 file2* prints only the lines common to the two files; **comm −23** *file1 file2* prints only lines in file1 but not in the file2; **comm −123** *file1 file2* is a no-op.

SEE ALSO

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

SUPPORT STATUS

Supported.

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 and replaced by *file1*.

If *mv* or *ln* determines that the mode of *target* forbids writing, it prints the mode (see *chmod*(1)), asks for a response, and reads the standard input for one line (if the standard input is a terminal). If the line begins with y, the move or link occurs; if not, the command exits.

Only *mv* allows *file1* to be a directory, in which case the directory rename occurs 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 save text bit is not set unless you are superuser; 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 is made. If *target* is a link to a file, all links remain and the file is changed.

OPTION

−f If the mode forbids writing, no questions are asked and the *mv* or *ln* is done when the −f option is used or if the standard input is not a terminal.

EXAMPLE

To copy all files in the *project* directory to the */mnt* directory, enter:

```
cp project/* /mnt
```

SEE ALSO

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

RESTRICTIONS

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

Ln does not link across file systems.

SUPPORT STATUS

Supported.

NAME

cpio − copy file archives in and out

SYNOPSIS

cpio −o [aBcTv [V pathname]]

cpio −i [bBcdfmrsStTuv6 [V pathname]] [patterns]

cpio −p [adlmruv] directory

DESCRIPTION

Cpio copies file archives and is typically used to make offline storage or backup copies of a directory and its files. *Cpio* **−o** (copy out) and *cpio* **−i** (copy in) are valid for streaming tape. *Cpio* **−p** (pass) is not valid for streaming tape.

Cpio **−o** (copy out) reads the standard input to obtain a list of pathnames and copies those files onto the standard output together with pathname and status information. Output is padded to a 512-byte boundary.

Cpio **−i** (copy in) extracts from the standard input (which is assumed to be the product of a previous *cpio* **−o**) the names of files selected by zero or more *patterns* given in the name-generating notation of *sh*(1). In *patterns*, metacharacters **?**, *****, and **[...]** match the slash **/** character. Multiple *patterns* may be specified. The default for *patterns* is ***** which selects all files. The extracted files are conditionally copied into the current directory tree based upon the chosen options. The permissions of the files are those of the previous *cpio* **−o**. The owner and group of the files is that of the current user unless the user is the superuser. If the user is the superuser, *cpio* retains the owner and group of the files of the previous *cpio* **−o**.

Cpio **−p** (pass) reads the standard input to obtain a list of pathnames of files that are conditionally created and copied into the destination *directory* tree based on the chosen options.

OPTIONS

- **a** Reset access times of input files after they are copied.
- **b** Swap both bytes and halfwords. Use only with the **−i** option.
- **B** Block input/output 5,120 bytes to the record (does not apply to the **−p** option; meaningful only with data directed to or from streaming tape.
- **c** Write *header* information in ASCII character form for portability, or read files created with *cpio* using the **−c** option.
- **d** Create *directories* as needed.
- **f** Copy in all files except those in *patterns*.
- **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.
- **r** Interactively *rename* files. If the user responds with a null line, the file is skipped.
- **s** Swap bytes. Use only with the **−i** option.
- **S** Swap halfwords. Use only with the **−i** option.

t Print a *table of contents* of the input; do not create any files.

T Provide reliable reporting of tape errors excluding end of file conditions. This option is only valid for streaming tape and is used only with the V option with −i or −o. This option overrides the B option and provides block I/O of 60K bytes per record.

u Copy *unconditionally* (normally, an older file does not replace a newer file with the same name).

v Print a list of file names. When used with the t option, the table of contents looks like the output of an ls −l command (see *ls*(1)).

V The next argument specifies the pathname of a file or device. Used for multi-tape or multi-volume copies with −i or −o. This option overrides the −B option and provides block I/O of 60K bytes per record.

6 Process an old (i.e., UNIX Sixth Edition format) file. Only useful with −i.

EXAMPLES

To copy all files in directory *thompson* to a streaming tape, enter

```
cd thompson
find . −print | cpio −ocvB > /dev/rstp/0yy
```

To copy all files on a streaming tape created by a *cpio* −o back to the directory *thompson*, enter

```
cd thompson
cpio −icvdB < /dev/rstp/0yy
```

To copy the contents of a directory into an archive, enter

```
ls | cpio −o > /dev/rstp/0yy
```

SEE ALSO

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

RESTRICTIONS

Path names are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the superuser can copy special files.

The −i and −o options are valid for streaming tape. The −p option is not valid for streaming tape.

Cpio requires /bin/mkdir and /bin/pwd; if these programs are not present, *cpio* pauses with no notification.

SUPPORT STATUS

Supported.

NAME

cpp − the C language preprocessor

SYNOPSIS

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

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc*(1) command. Thus the output of *cpp* is designed to be in a form acceptable as input to the next pass of the C compiler. The use of *cpp* other than in this framework is not recommended. See *m4*(1) for a general macro processor.

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

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

OPTIONS

The following options to *cpp* are recognized:

−P Preprocess the input without producing the line control information used by the next pass of the C compiler.

−C By default, *cpp* strips C-style comments. If the **−C** option is specified, all comments (except those found on *cpp* directive lines) are passed along.

−U*name*

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

operating system:	ibm, gcos, os, tss, unix
hardware:	tower, interdata, pdp11, u370, u3b, u3b5, vax
UNIX system variant:	RES, RT
lint(1):	lint

−D*name*

−D*name*=*def*

Define *name* as if by a **#define** directive. If no =*def* is given, *name* is defined as 1. The **−D** option has lower precedence than the **−U** option. That is, if the same name is used in both a **−U** option and a **−D** option, the name is undefined regardless of the order of the options.

−T Preprocessor symbols are no longer restricted to eight characters. The **−T** option forces *cpp* to use only the first eight characters for distinguishing different preprocessor names. This behavior is the same as previous preprocessors with respect to the length of names and is included for backward compatability.

−I*dir*

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

DIRECTIVES

All *cpp* directives start with lines begun by #. Any number of blanks and tabs are allowed between the # and the directive. The directives are:

#define *name token-string*

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

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

Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma-separated set of tokens, and a) by *token-string*, where each occurrence of an *arg* in the *token-string* is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* is expanded, *cpp* re-starts its scan for names to expand at the beginning of newly created *token-string*.

#undef *name*

Remove the definition of *name* (if any).

#include "*filename*"
#include <*filename*>

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

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

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

#endif

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

#ifdef *name*

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

#ifndef *name*

The lines following do not appear in the output if and only if *name* is the subject of a previous **#define** without being the

subject of an intervening **#undef**.

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

#else Reverse the notion of the test directive which matches this directive. So if lines previous to this directive are ignored, the following lines appear in the output.

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

DIRECTORY SEARCH

The directory search order for **#include** files is:

1. Directory of the file which contains the **#include** request
2. Directories specified by **−I** in left to right order
3. Standard directory */usr/include*

CONVENTIONS

Linefeeds:
An unescaped linefeed (the single character \n) terminates a character constant or quoted string. An escaped linefeed (the two character sequence \\\n) may be used in the body of a **#define** statement to continue the definition into the next line. The escaped linefeed is converted into a single blank in the macro body.

Comments:
Comments are uniformly removed unless the **−C** is specified. Comments are also ignored except that a comment terminates a token. Thus "off/* comment */bar" may expand *off* and *bar* but never expands *offbar*. **If neither** *off* **nor** *bar* is a macro, the output is *offbar* even if *offbar* is defined as something else. The file:

```
#define off(a,b)b/**/a
off(1,2)
```

produces 21 because the comment causes a break which enables the recognition of *b* and *a* as formals in the string "b/**/a".

Macro Processing:
Macro formal parameters are recognized in **#define** bodies even inside character constants and quoted strings. The output from:

```
#define off(a) '\a'
off(bar)
```

is the six characters "'\bar'". Macro names are not recognized inside character constants or quoted strings during the regular scan. Thus:

```
#define off bar
printf("off");
```

does not expand *off* in the second line because it is inside a quoted string which is not part of a **#define** macro definition.

Macros are not expanded while processing a **#define** or **#undef**. Thus:

```
#define off invalid
#define bar off
#undef off
bar
```

produces *off*. The token immediately after an **#ifdef** or **#ifndef** is not expanded.

Macros are not expanded during the scan which determines the actual parameters to another macro call. When a macro is expanded, the first step is to put the actual arguments in the corresponding locations in the token string the macro is defined to be. The next step is to start to reprocess the token string as input text.

If the **—R** option is not specified, the invocation of some recursive macros is trapped and the recursion is forcibly terminated with an error message. The trapped recursions are those in which the nesting level is non-decreasing from some point on. In particular,

```
#define a a
a
```

is detected. Consider using "#undef a". The recursion:

```
#define a c b
#define b c a
#define c off
a
```

is not detected because the nesting level decreases after each expansion of *c*.

The **—R** option specifically allows recursive macros and recursion is strictly obeyed to the extent that space is available. Assuming that **—R** is specified:

```
#define a a
a
```

causes an infinite loop with very little output. The tail recursion

```
#define a <b
#define b >a
a
```

causes the string "<>" to be output infinitely many times. The non-tail recursion

```
#define a b>
#define b a<
a
```

complains "too much pushback", dumps the pushback, and continues infinitely.

"1.e4" is recognized as a floating point number and not as a request to expand the macro name "e4".

Any kind and amount of white space (space, tab, linefeed, vertical tab, formfeed, or newline) is allowed between a macro name and the left parenthesis which introduces its actual parameters.

The comma operator, macros with parameters, single-character character constants, and comments are valid in preprocessor **#if** statements.

Linefeeds are output in the correct place when a multiline comment is not passed through to the output.

Nothing, including linefeeds, is output while a false **#if**, **#ifdef**, or **#ifndef** is in effect. Thus when all conditions become true, a line of the form

```
# 12345 \"off.c\""
```

is output unless the —**P** option is specified.

Error and warning messages are output on standard error.

Mismatch between the number of formal and actual parameters in a macro call produces only a warning and not an error. Excess actual parameters are ignored; missing actual parameters are turned into null strings.

Newlines found during the scan for actual arguments are changed to blanks. Formfeeds act like newlines with respect to recognizing # as the flag for *cpp*.

FILES

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

SEE ALSO

cc(1), m4(1).

DIAGNOSTICS

The error messages produced by *cpp* are self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When new-line characters were found in argument lists for macros to be expanded, previous versions of *cpp* put out the new-lines as they were found and expanded. The current version of *cpp* replaces these new-lines with blanks to alleviate problems that the previous versions had when this occurred.

SUPPORT STATUS

Supported.

NAME

crontab − user crontab file

SYNOPSIS

crontab [file]
crontab −r
crontab −l

DESCRIPTION

Crontab copies the specified file, or standard input if no file is specified, into a directory that holds all user crontab files. A user crontab file specifies commands to be executed at specific dates and times by *cron*(1M).

A user is permitted to use *crontab* if their name appears 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. 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 valid 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**. Because *login*(1) is not invoked, **.profile** is not automatically executed. 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)**.

NOTE: Users should remember to redirect the standard output and standard error of their commands. If this is not done, any generated output or errors is mailed to the user.

OPTIONS

−r Removes a user crontab from the crontab directory.

−l Lists the crontab file for the invoking user.

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

login(1), sh(1), cron(1M).

SUPPORT STATUS

Supported.

NAME

crypt − encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

Crypt reads from the standard input and writes on the standard output. *Password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *Crypt* encrypts and decrypts with the same key:

```
crypt key <file1>file2
crypt key <file2 | pr
```

will print the file file1.

Files encrypted by *crypt* are compatible with those treated by the editor *ed* in encryption mode.

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

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e. it takes a substantial fraction of a second to compute. However, if keys are restricted to, for example, three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. The choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES

/dev/tty	for typed key

SEE ALSO

ed(1), makekey(1).

RESTRICTIONS

If output is piped to *nroff* and the encryption key is *not* given on the command line, *crypt* can leave terminal modes in a strange state (see *stty*(1)).

If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.

SUPPORT STATUS

Supported. *Crypt* is available only in the United States.

NAME

csh − a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [−cefinstvVxX] [arg ...]

DESCRIPTION

Csh is a command language interpreter incorporating a history mechanism and a C-like syntax.

An instance of *csh* begins by executing commands from the file *.cshrc* in the *home* directory of the invoker. If this is a login shell, *csh* also executes commands from the file *.login* there. It is typical for users on terminals to put the command **stty crt** in their *.login* file.

In the normal case, the shell then begins reading commands from the terminal, prompting with %. Processing of arguments and the use of the shell to process files containing command scripts is described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and parsed. Then each command in the current line is executed.

When a login shell terminates, it executes commands from the file *.logout* in the user home directory.

LEXICAL STRUCTURE

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters &, |, ;, <, >, (, and) form separate words. If doubled in single quotation marks, as '&&', '||', '<<' or '>>' these pairs form single words. These parser metacharacters may be made part of other words, or their special meaning may be prevented, by preceding them with a backslash, \. A newline preceded by a \ is equivalent to a blank. It is usually necessary to use the backslash to escape the parser metacharacters when you want to use them literally rather than as metacharacters.

Strings enclosed in matched pairs of quotation marks, either single or double quotation marks, form parts of a word. Metacharacters in these strings, including blanks and tabs, do not form separate words. Such quotations have semantics to be described subsequently.

Within pairs of single or double quotation marks a newline (carriage return) preceded by a \ gives a true newline character. This is used to set up a file of strings separated by newlines, as for *fgrep(1)*.

When the shell input is not a terminal, the character # introduces a comment which continues to the end of the input line. It is prevented from having this special meaning when preceded by \ or if bracketed by a pair of single or double quotation marks.

COMMANDS

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a

sequence of simple commands separated by | characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by semicolons (;), and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an &, which means to run it in background.

Parentheses (and) around a pipeline or sequence of pipelines cause the whole series to be treated as a simple command, which may in turn be a component of a pipeline, etc. It is also possible to separate pipelines with '| |' or '&&' indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See **Expressions.**)

PROCESS I.D. NUMBERS

When a process is run in background with &, the shell prints a line which looks like:

1234

indicating that the process which was started asynchronously was number 1234.

STATUS REPORTING

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work.

To check on the status of a process, use the *ps* (process status) command.

SUBSTITUTIONS

The shell performs various transformations on the input; these transformations are described in the order in which they occur.

History Substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence.

History substitutions begin with the character ! and may begin anywhere in the input stream with the provision that they do not nest.

This ! may be preceded by a \ to turn off its special meaning; for convenience, a ! is also passed unchanged when it is followed by a blank, tab, newline, = or (.

Therefore, do not put a space after the ! and the command reference when invoking the history mechanism. History substitutions also occur when an input line begins with ↑. This special abbreviation is described later.

An input line which invokes history substitution is echoed on the terminal before it is executed, as it would look if typed out in full.

The shell history list, which may be seen by typing the history command, contains all commands input from the terminal which consist of one or more words. History substitutions reintroduce sequences of words from these saved commands into the input stream. The history variable controls the size of the input stream. The previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

Consider the following output from the history command:

```
 9 write michael
10 ex write.c
11 cat oldwrite.c
12 diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the prompt by placing an ! in the prompt string. This is done by using the set command to set the Prompt = ! and the prompt character of your choice.

For example, if the current event is number 13, the command recorded as event 11 can be called in several ways:

- as !-2 [i.e., 13-2];
- by the first letter of one of its command words, such as !c referring to the c in *cat;*
- or !wri for event 9, or by a string contained in a word in the command as in !?mic? also referring to event 9.

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case '!!' refers to the previous command; thus '!!' alone is essentially a redo.

Words are selected from a command event and acted upon according to the following formula:

```
event:position:action
```

The event is the command you wish to retrieve. As mentioned above, it may be summoned up by event number and in several other ways. All that the event notation does is to tell the shell which command you want.

Position picks out the words from the command event on which you want the action to take place. The position notation can do anything from altering the command completely to making some very minor substitution, depending on which words from the command event you specify with the position notation.

To select words from a command event, follow the event specification with a : and a designator by position for the desired words.

The words of a command event are picked out by their position in the input line. Positions are numbered from 0, the first word (usually command) being position 0, the second word having position 1,

and so forth. If you designate a word from the command event by stating its position, that means you want to include it in your revised command. All the words that you want to include in a revised command must be designated by position notation in order to be included.

The basic position designators are:

0	first (command) word
n	*n* 'th argument
↑	first argument, i.e. 1
$	last argument
%	matches the word of an ?s? search which immediately precedes it; used to strip one word out of a command event for use in another command. Example: !?four?:%:p prints four.
x−*y*	range of words (e.g. 1−3 means from position 1 to position 3).
−*y*	abbreviates 0−*y*
*	stands for ↑−$, or indicates position 1 if only one word in event.
*x**	abbreviates *x*−$ where x is a position number.
x−	like *x** but omitting last word $

The : separating the event specification from the word designator can be omitted if the argument selector begins with a ↑, $, *, −, or %.

Modifiers, each preceded by a colon (:), may be used to act on the designated words in the specified command event. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing .xxx component, leaving the root name.
e	Remove all but the extension .xxx part.
s/*old*/*new*/	Substitute *new* for *old*
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g. g&.
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

Unless preceded by a g, the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

The left side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of /; a \ quotes the delimiter into the *l* and *r* strings. The character & in the right side is replaced by the text from the left. A \ quotes & also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in !?*s*?. The trailing delimiter in the substitution may be omitted if (but only if) a newline follows immediately as may the trailing ? in a contextual scan.

A history reference may be given without an event specification, e.g. !$. In this case the reference is to the previous command. If a previous history reference occurred on the same line, this form repeats the previous reference. Thus '!?foo?↑ !$' gives the first and last arguments from the command matching ?foo?.

You can quickly make substitutions to the previous command line by using the ↑ character as the first non-blank character of an input line. This is equivalent to !:s↑ providing a convenient shorthand for substitutions on the text of the previous line. Thus ↑lb↑lib fixes the spelling of lib in the previous command.

A history substitution may be surrounded with { and } if necessary to insulate it from the characters which follow. Thus, after 'ls −ld paul' you might do !{l}a to do 'ls −ld paula', while !la would look for a command starting la.

Quotations

The quotation of strings by single and double quotation marks can be used to prevent all or some of the remaining substitutions which would otherwise take place if these characters were interpreted as metacharacters or wild card matching characters. Strings enclosed in single quotation marks are prevented any further interpretation or expansion. Strings enclosed in double quotation marks may still be variable and command expanded as described below.

In both cases the resulting text becomes all or part of a single word; only in one special case (see **Command Substitution** below) does a string enclosed in double quotation marks yield parts of more than one word; strings enclosed in single quotation marks never do.

Alias Substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for 'ls' is 'ls −l' the command 'ls /usr' would map to "ls −l /usr", the argument list here being undisturbed. Similarly if the alias for 'lookup' was 'grep !↑ /etc/passwd' then "lookup bill" would map to "grep bill /etc/passwd".

If an alias is found, the word transformation of the input text is performed and the alias process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus you can 'alias print 'pr \!* | lpr'' to make a command which *pr*'s its arguments to the line printer.

Variable Substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the argv variable is an image of the shell argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the −v command line option.

Other operations treat variables numerically. The @ command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as zero or more strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by $ characters. This expansion can be prevented by preceding the $ with a \ except within strings enclosed in double quotation marks where it always occurs, and within strings enclosed in single quotation marks where it never occurs. Strings enclosed in single quotation marks are interpreted later (see **Command Substitution** below) so $ substitution does not occur there until later, if at all. A $ is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in double quotation marks or given the :q modifier the results of variable substitution may eventually be command and filename substituted. Within double quotation marks a variable whose value consists of multiple words expands to a portion of a single word, with the words of the variables value separated by blanks. When the :q modifier is applied to a substitution the variable expands to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

Metasequences for Variable Substitution

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$name
${name}
: Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following

characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter. If *name* is not a shell variable, but is set in the environment, then that value is returned but : modifiers and the other forms given below are not available in this case.

$name[selector]
${name[selector]}
: May be used to select only some of the words from the value of *name*. The selector is subjected to $ substitution and may consist of a single number or two numbers separated by a −. The first word of a variables value is numbered 1. If the first number of a range is omitted it defaults to 1. If the last member of a range is omitted it defaults to $#name. The selector * selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}
: Gives the number of words in the variable. This is useful for later use in a [selector].

$0 Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$number
${number}
: Equivalent to $argv[number].

$* Equivalent to $argv[*].

The modifiers :h, :t, :r, :q, and :x may be applied to the substitutions above as may :gh, :gt, and :gr. If braces appear in the command form then the modifiers must appear within the braces. Only one : modifier is allowed on each $ expansion.

The following substitutions may not be modified with : modifiers.

$?name
${?name}
: Substitutes the string 1 if name is set, 0 if it is not.

$?0 Substitutes 1 if the current input filename is known, 0 if it is not.

$$ Substitutes the (decimal) process number of the (parent) shell.

$< Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Command and Filename Substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command Substitution

Command substitution is indicated by a command enclosed in single quotation marks. The output from such a command is

normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within double quotation marks, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename Substitution

If a word contains any of the characters *, ?, [, or { or begins with the character , then that word is a candidate for filename substitution, also known as globbing. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters *, ?, and [imply pattern matching, the characters and { being more akin to abbreviations.

In matching filenames, the character period (.) at the beginning of a filename or immediately following a /, as well as the character / must be matched explicitly. The character * matches any string of characters, including the null string. The character ? matches any single character. The sequence [...] matches any one of the characters enclosed. Within [...], a pair of characters separated by - matches any character lexically between the two.

The character at the beginning of a filename is used to refer to home directories. Standing alone, it expands to the invokers home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits, and - characters the shell searches for a user with that name and substitutes their home directory; thus ken might expand to /usr/ken and ken/chmach to /usr/ken/chmach. If the character is followed by a character other than a letter or / or appears not at the beginning of a word, it is left undisturbed.

The metanotation a{b,c,d}e is a shorthand for 'abe ace ade'. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus source/s1/{oldls,ls}.c expands to /usr/source/s1/oldls.c /usr/source/s1/ls.c whether or not these files exist without any chance of error if the home directory for source is /usr/source. Similarly ../{memo,*box} might expand to ../memo ../box ../mbox. Note that memo was not sorted with the results of matching *box. As a special case {, }, and {} are passed undisturbed.

Input/Output

The standard input and standard output of a command may be redirected with the following syntax:

< name

Open file *name* (which is first variable, command and filename expanded) as the standard input.

<< word

Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting \, ", ' (accent), or ` (grave) appears in *word*, variable and command substitution is performed on the intervening lines, allowing \ to quote $, \, and `. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name

>! name

>& name

>&! name

The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, it is truncated, its previous contents being lost. If the variable *noclobber* is set, then the file must not exist or be a character special file (e.g. a terminal or /dev/null) or an error results. This helps prevent accidental destruction of files. In this case the ! forms can be used and suppress this check. The forms involving & route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in the same way as < input filenames are.

>> name

>>& name

>>! name

>>&! name

Uses file *name* as standard output like > but places output at the end of the file. If the variable *noclobber* is set, then it is an error for the file not to exist unless one of the ! forms is given. Otherwise similar to >.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The '<<' mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input.

Diagnostic output may be directed through a pipe with the standard output. Simply use the form |& rather than just |.

Expressions

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the *@*, *exit*, *if*, and *while* commands. The following operators are available:

|| && | ↑ & == != = ! <= >= < > << >> + - * / % ! ( )

Here the precedence increases to the right, ==, !=, = , and ! ; <=, >=, <, and >; << and >>; + and -; *, /, and % being, in groups, at the same level. The ==, !=, = , and ! operators compare their arguments as strings; all others operate on numbers. The operators = and ! are like != and == except that the right hand side is a pattern (containing, e.g. *s, ?s and instances of [...]) against which the left operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with 0 are considered octal numbers. Null or missing arguments are considered 0. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser- &, |, <, >, (, and) - they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in { and } and file enquiries of the form '-*l* name' where *l* is one of:

r	read access	w	write access
x	execute access	e	existence
o	ownership	z	zero size
f	ordinary file	d	directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible, then all enquiries return false, i.e. 0. Command executions succeed, returning true, i.e. 1, if the command exits with status 0, otherwise they fail, returning false, i.e. 0. If more detailed status information is required then the command should be executed outside of an expression and the variable *status* examined.

CONTROL FLOW

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach, switch,* and *while* statements, as well as the *if-then-else* form of the *if* statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. To the extent that this allows, backward goto statements succeed on non-seekable inputs.

BUILTIN COMMANDS

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last then it is executed in a subshell.

alias
alias name
alias name wordlist
: The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name; wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias.*

break
: Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while.* The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw
: Causes a break from a *switch,* resuming after the *endsw.*

case label:
: A label in a *switch* statement as discussed below.

cd
cd name
chdir
chdir name
: Change the shell working directory to directory *name.* If no argument is given then change to the home directory of the user. If *name* is not found as a subdirectory of the current directory and does not begin with /, ./, or ../, then each component of the variable *cdpath* is checked to see if it has a subdirectory *name.* If all else fails but *name* is a shell variable whose value begins with /, then this is tried to see if it is a directory.

continue
: Continue execution of the nearest enclosing *while* or *foreach.* The rest of the commands on the current line are executed.

default:
: Labels the default case in a *switch* statement. The default should come after all *case* labels.

echo wordlist
echo −**n** wordlist
: The specified words are written to the shell standard output, separated by spaces, and terminated with a newline unless the −**n** option is specified.

else
end
endif
endsw
: See the description of the *foreach, if, switch,* and *while* statements.

exec command
: The specified command is executed in place of the current shell.

exit
exit(expr)
The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

foreach name (wordlist)
...
end The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.) The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read once prompting with ? before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

glob wordlist
Like *echo* but no \ escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto word
The specified *word* is filename and command expanded to yield a string of the form label. The shell rewinds its input as much as possible and searches for a line of the form label: possibly preceded by blanks or tabs. Execution continues after the specified line.

history
Displays the history event list.

if (expr) command
If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is not executed.

if (expr) **then**
...
else if (expr2) **then**
...
else
...
endif
If the specified *expr* is true then the commands to the first *else* are executed; else if *expr2* is true then the commands to the second else are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear at the beginning of its input line or after an *else*.

kill pid
kill −sig pid ...
kill −l

Sends either the TERM (terminate) signal or the specified signal to the specified processes. Signals are either given by number or by names (as given in */usr/include/signal.h*, stripped of the prefix SIG). The signal names are listed by kill −l. There is no default, saying just kill does not send a signal to the current process. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process is sent a CONT (continue) signal as well.

login

Terminate a login shell, replacing it with an instance of **/bin/login**. This is one way to log off, included for compatibility with *sh*(1).

logout

Terminate a login shell. Especially useful if *ignoreeof* is set.

nice
nice +number
nice command
nice +number command

The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number. The final two forms run command at priority 4 and *number* respectively. The superuser may specify negative niceness by using 'nice −number ...'. Command is always executed in a sub-shell, and the restrictions placed on commands in simple *if* statements apply.

nohup
nohup command

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with & are effectively *nohup*'ed.

onintr
onintr −
onintr label

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form 'onintr −' causes all interrupts to be ignored. The final form causes the shell to execute a 'goto label' when an interrupt is received or a child process terminates because it was interrupted. In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

rehash

Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while

you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command
The specified *command*, which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set
set name
set name=word
set name[index]=word
set name=(wordlist)
The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index'th* component of name to word; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded. These arguments may be repeated to set multiple values in a single set command. Note, however, that variable expansion happens for all arguments before any setting occurs.

setenv name value
Sets the value of environment variable *name* to be *value*, a single string. The variable PATH is automatically imported to and exported from the *csh* variable *path;* there is no need to use *setenv* for these.

shift
shift variable
The members of *argv* are shifted to the left, discarding *argv[1]*. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source name
The shell reads commands from *name*. *Source* commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Input during *source* commands is never placed on the history list.

switch (string)
case str1:
...
breaksw
...
default:
...
breaksw
endsw
Each case label is successively matched against the specified *string* which is first command and filename expanded. The

file metacharacters *, ?, and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a default label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

time
time command
With no argument, a summary of time used by this shell and its children is printed. If arguments are given the specified simple command is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

umask
umask value
The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern
All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias *'. It is not an error for nothing to be *unaliased*.

unhash
Use of the internal hash table to speed location of executed programs is disabled.

unset pattern
All variables whose names match the specified pattern are removed. Thus all variables are removed by 'unset *'; this has noticeably bad side-effects. It is not an error for nothing to be *unset*.

wait All background jobs are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)
...
end While the specified expression evaluates non-zero, the commands between the *while* and the matching end are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. The *while* and *end* must appear alone on their input lines. Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

@
@ name = expr
@ name[index] = expr
The first form prints the values of all the shell variables. The

second form sets the specified *name* to the value of *expr.* If the expression contains <, >, &, or | then at least this part of the expression must be placed within (and). The third form assigns the value of *expr* to the *index'th* argument of *name.* Both *name* and its *index'th* component must already exist. The operators *=, +=, etc. are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* which would otherwise be single words. Special postfix ++ and −− operators increment and decrement *name* respectively, i.e. @ i++.

PRE-DEFINED AND ENVIRONMENT VARIABLES

The following variables have special meaning to the shell. Of these, *argv, home, path, prompt, shell* and *status* are always set by the shell. Except for *status* this setting occurs only at initialization; these variables are not then modified unless this is done explicitly by the user.

This shell copies the environment variable USER into the variable *user,* TERM into *term,* and HOME into *home,* and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file *.cshrc* as interior *csh* processes import the definition of *path* from the environment, and re-export it if you then change it.

argv Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e. $1 is replaced by $argv[1], etc.

cdpath
Gives a list of alternate directories searched to find subdirectories in *chdir* commands.

echo Set when the −x command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, because these substitutions are then done selectively.

history
Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events is not discarded. Too large values of *history* may run the shell out of memory. The last executed command is always saved on the history list.

home
The home directory of the invoker, initialized from the environment. The filename expansion of refers to this variable.

ignoreeof
If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-d.

mail The files where the shell checks for mail. This is done after each command completion which results in a prompt, if a specified interval has elapsed. The shell says 'You have new mail.' if the file exists with an access time not greater than its modify time. If the first word of the value of *mail* is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes. If multiple mail files are specified, then the shell says says 'New mail in *name*' when there is mail in the file *name*.

noclobber
As described in the section on **Input/Output**, restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that >> redirections refer to existing files.

noglob
If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch
If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. "echo [" still gives an error.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable then only full path names execute. The usual search path is ., /bin, and /usr/bin, but this may vary from system to system. For the superuser the default search path is /etc, /bin, and /usr/bin. A shell which is given neither the −c nor the −t option normally hashes the contents of the directories in the *path* variable after reading *.cshrc*, and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash* or the commands may not be found.

prompt
The string which is printed before each command is read from an interactive terminal input. If a ! appears in the string it is replaced by the current event number unless a preceding \ is given. Default is % or # for the superuser.

shell The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of **Non-builtin Command Execution** below.) Initialized to the pathname of the shell.

status
The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status 1, all other builtin commands set status 0.

time Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds

causes a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.

verbose
Set by the −v command line option, causes the words of each command to be printed after history substitution.

NON-BUILTIN COMMAND EXECUTION

When a command to be executed is found not to be a builtin command the shell attempts to execute the command via *exec*(2). Each word in the variable *path* names a directory from which the shell attempts to execute the command. If it is given neither a −c nor a −t option, the shell hashes the names in these directories into an internal table so that it only tries an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash*), or if the shell was given a −c or −t option, and in any case for each directory component of *path* which does not begin with a /, the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus "(cd ; pwd) ; pwd" prints the *home* directory; leaving you where you were (printing this after the home directory), while "cd ; pwd" leaves you in the *home* directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias are prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (e.g. $shell). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

ARGUMENT LIST PROCESSING

If argument 0 to the shell is - then this is a login shell. The flag arguments are interpreted as follows:

−c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

−e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

−f The shell starts faster because it neither searches for nor executes commands from the file *.cshrc* in the invokers home directory.

−i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

−n Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.
−s Command input is taken from the standard input.
−t A single line of input is read and executed. A \ may be used to escape the newline at the end of this line and continue onto another line.
−v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.
−x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.
−V Causes the *verbose* variable to be set even before *.cshrc* is executed.
−X Is to −x as −V is to −v.

After processing of flag arguments, if arguments remain but none of the −c, −i, −s, or −t options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by $0. Remaining arguments initialize the variable *argv*.

SIGNAL HANDLING

The shell normally ignores *quit* signals. Processes running in background (by &) are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell parent. In no case are interrupts allowed when a login shell is reading the file *.logout*.

FILES

/.cshrc	Read at beginning of execution by each shell.
/.login	Read by login shell, after *.cshrc* at login.
/.logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a #.
/tmp/sh*	Temporary file for '<<'.
/etc/passwd	Source of home directories for name.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 5120 characters. The number of arguments to a command which involves filename expansion is limited to 1/6th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

sh(1), access(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(4).

RESTRICTIONS

It suffices to place the sequence of commands in parentheses to force it to a subshell, i.e. (a ; b ; c).

Control over tty output after processes are started is primitive.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by ?, are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. All and more than one : modifier should be allowed on $ substitutions.

SUPPORT STATUS

Supported.

NAME

csplit − context split

SYNOPSIS

csplit [−s] [−k] [−f prefix] file arg1 [... argn]

DESCRIPTION

Csplit reads *file* and separates it into n+1 sections, defined by the arguments *arg1... argn*. The sections are placed in xx00 ... xx*n* where *n* is less than 100). 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 − then standard input is used.

As *csplit* creates the files, it prints the character counts for each file created. If an error occurs, *csplit* removes the created files. *Csplit* does not affect the original file.

OPTIONS

−s Suppresses the printing of all character counts.

−k Leaves previously created files intact if an error occurs.

−f *prefix* Names the created files *prefix*00 ... *prefixn* rather than xx0 ... xx*n*.

The arguments *arg1 ... argn* denote line references and can be a combination of the following:

/rexp/ Creates a file 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% Works the same as */rexp/*, except that no file is created for the section.

lineno Creates a file from the current line up to (but not including) *lineno*. The current line becomes *lineno*.

{num} Repeats the preceding argument *num* times. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lineno*, the file is split every *lineno* lines (*num* times) from that point.

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the Shell in the appropriate quotes. Regular expressions may not contain embedded new-lines.

EXAMPLES

This example creates four files, **cobol00 . . . cobol03.**

```
csplit −f cobol cobolfile '/procedure division/' /par5./ /par16./
```

After editing the split files, they can be recombined as follows:

```
cat cobol0[0−3] > cobolfile
```

Note that this example overwrites the original file.

```
csplit −k fortfile 100 {99}
```

This example would split fortfile 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 creates a file containing each separate C routine (up to 21) in **prog.c.**

SEE ALSO

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

DIAGNOSTICS

Self explanatory except for:

arg − out of range

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

SUPPORT STATUS

Supported.

NAME

ct − spawn getty to a remote terminal

SYNOPSIS

ct [−h] [−v] [−wn] [−sspeed] telno ...

DESCRIPTION

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

Ct tries each line listed in the file **/usr/lib/uucp/L-devices** until it finds an available line with appropriate attributes or runs out of entries. If there are no free lines, *ct* asks if it should wait for one, and if so, for how many minutes it should wait before it gives up. *Ct* continues to try to open the dialers at one-minute intervals until the specified limit is exceeded.

Normally, *ct* hangs up the current line if a line registers an incoming call.

Transmission speed is set by the **−s** option, and the default rate is 300 baud.

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

Note that the destination terminal must be attached to a modem that can answer the telephone.

OPTIONS

−w*n* Denotes **n** as the maximum number of minutes that *ct* is to wait for a line, overriding the usual dialogue

−h Retains the line if an incoming call is registered.

−v Sends a running narrative to the standard error output stream.

−s*speed*
Sets transmission speed to *speed* where *speed* is a baud rate.

FILES

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

SEE ALSO

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

SUPPORT STATUS

Supported.

NAME

ctags − create a tags file

SYNOPSIS

ctags [−BFatuwx] name ...

DESCRIPTION

Ctags makes a tags file for *ex*(1) from the specified C, Pascal, Fortran, yacc, lex, and lisp sources. A tags file gives the locations of specified objects (in this case functions and typedefs) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern, typedefs with a line number. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the *tags* file, *ex* can quickly find these object definitions.

Files whose name ends in .c or .h are assumed to be C source files and are searched for C routine and macro definitions. Files whose names end in .y are assumed to be yacc source files. Files whose names end in .l or if their first non-blank character is ;, (, or [are assumed to be lisp files. Files whose names end in .l and if their first non-blank character is not ;, (, or [are assumed to be lex files. Others are first examined to see if they contain any Pascal or Fortran routine definitions; if not, they are processed again looking for C definitions.

The tag *main* is treated specially in C programs. The tag formed is created by prepending *M* to the name of the file, with a trailing .c removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

OPTIONS

−x Produce a list of object names, the line number and file name on which each is defined, as well as the text of that line and print this on the standard output. This is a simple index which can be printed as an off-line readable function index.

−F Use forward searching patterns (/.../) (default).

−B Use backward searching patterns (?...?).

−a Append to tags file.

−t Create tags for typedefs.

−w Suppress warning diagnostics.

−u Cause the specified files to be *updated* in tags: delete all references to the files, and append the new values to the file. (Beware: the implementation of this option is rather slow; it is usually faster to simply rebuild the *tags* file.)

FILES

tags	output tags file

SEE ALSO

ex(1), vi(1).

RESTRICTIONS

Recognition of **functions, subroutines** and **procedures** for FORTRAN and Pascal is implemented simplistically. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name, *ctags* does not function properly.

Ctags does not process #ifdefs.

Ctags should know about Pascal types. It relies on the input being well formed to detect typedefs.

Use of −tx shows only the last line of typedefs.

SUPPORT STATUS

Supported.

NAME

ctrace − C program debugger

SYNOPSIS

ctrace [options] [file]

DESCRIPTION

Ctrace allows you to follow the execution of a C program, statement by statement. The effect is similar to executing a shell procedure with the −x option. *Ctrace* reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of *ctrace* into a temporary file because the *cc*(1) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it is listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the *bfs*(1) or *tail*(1) command.

OPTIONS

The only *options* you commonly use are:

−**f** *functions* Trace only these *functions.*
−**v** *functions* Trace all but these *functions.*

You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are printed as strings if appropriate. Char, short, and int variables are printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. String arguments to the *string*(3C) functions and return values from *fgets*(3S), *gets*(3S), and *sprintf*(3S) are printed as strings. You can request that variables be printed in additional formats, if appropriate, with these *options*:

−**o** Octal
−**x** Hexadecimal
−**u** Unsigned
−**e** Floating point

These *options* are used only in special circumstances:

−**l** *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.
−**s** Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.

−t *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The Diagnostics section explains when to use this option.

−P Run the C preprocessor on the input before tracing it. You can also use the **−D**, **−I**, and **−U** *cc*(1) preprocessor options.

These *options* are used to tailor the run-time trace package when the traced program is run in a non-UNIX system environment:

−b Use only basic functions in the trace code, that is, those in *ctype*(3C), *printf*(3S), and *string*(3C). These are usually available. In particular, this option is needed when the traced program runs under an operating system that does not have *signal*(2), *fflush*(3S), *longjmp*(3C), or *setjmp*(3C).

−p '*s*' Change the trace print function from the default of 'printf('. For example, 'fprintf(stderr,' sends the trace to the standard error output.

−rf Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **−p** option).

EXAMPLE

If the file *lc.c* contains this C program:

```
 1 #include <stdio.h>
 2 main()        /* count lines in input */
 3 {
 4       int c, nl;
 5
 6       nl = 0;
 7       while ((c = getchar()) != EOF)
 8               if (c = '\n')
 9                       ++nl;
10       printf("%d\n", nl);
11 }
```

and you enter these commands and test data: cc lc.c a.out 1 (cntl-d), the program is compiled and executed. The output of the program is the number 2, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke *ctrace* with these commands: ctrace lc.c >temp.c cc temp.c a.out the output is:

```
 2 main()
 6       nl = 0;
         /* nl == 0 */
 7       while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output is:

```
         /* c == 49 or '1' */
 8               if (c = '\n')
```

```
                /* c == 10 or '\n' */
9                       ++nl;
                        /* nl == 1 */
7       while ((c = getchar()) != EOF)
        /* c == 10 or '\n' */
8               if (c = '\n')
                /* c == 10 or '\n' */
9                       ++nl;
                        /* nl == 2 */
7       while ((c = getchar()) != EOF)
```

If you now enter an end of file character (cntl-d) the final output is:

```
        /* c == -1 */
10      printf("%d\n", nl);
        /* nl == 2 */2
         return
```

Note that the program output is printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you probably realize that you used the assignment operator (=) in place of the equal operator (==). You can easily miss this error during code reading.

EXECUTION-TIME TRACE CONTROL

The default operation for *ctrace* is to trace the entire program file, unless you use the −f or −v option to trace specific functions. This does not give you statement by statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding *ctroff*() and *ctron*() function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with *if* statements, and you can even conditionally include this code because *ctrace* defines the **CTRACE** preprocessor variable. For example:

```
#ifdef CTRACE
        if (c == '!' && i > 1000)
                ctron();
#endif
```

You can also call these functions from *sdb*(1) if you compile with the −g option. For example, to trace all but lines 7 to 10 in the main function, enter:

```
sdb a.out
main:7b ctroff()
main:11b ctron()
```

r

You can also turn the trace off and on by setting static variable tr_ct_ to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly, such as *adb*(1).

DIAGNOSTICS

This section contains diagnostic messages from both *ctrace* and *cc*(1), because the traced code often gets some *cc* warning messages. You can get *cc* error messages in some rare cases, all of which can be avoided.

Ctrace Diagnostics

warning: some variables are not traced in this statement
Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the −t option to increase this number.

warning: statement too long to trace
This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

cannot handle preprocessor code, use −P option
This is usually caused by #ifdef/#endif preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

'if ... else if' sequence too long
Split the sequence by removing an else from the middle.

possible syntax error, try −P option
Use the −P option to preprocess the *ctrace* input, along with any appropriate −D, −I, and −U preprocessor options. If you still get the error message, check the Warnings section below.

Cc Diagnostics

warning: floating point not implemented
warning: illegal combination of pointer and integer
warning: statement not reached
warning: sizeof returns 0
Ignore these messages.

compiler takes size of function
See the *ctrace* "possible syntax error" message above.

yacc stack overflow
See the *ctrace* "'if ... else if' sequence too long" message above.

out of tree space; simplify expression
Use the −t option to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you now get.

redeclaration of signal
Either correct this declaration of *signal*(2), or remove it and #include <signal.h>.

WARNINGS

You get a *ctrace* syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Use a different name.

Ctrace assumes that BADMAG is a preprocessor macro, and that EOF and NULL are #defined constants. Declaring any of these to be variables, e.g. "int EOF;", causes a syntax error.

RESTRICTIONS

Ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. *Ctrace* may choose to print the address of an aggregate or use the wrong format (e.g., %e for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

FILES

runtime.c	run-time trace package

SEE ALSO

signal(2), ctype(3C), fflush(3S), longjmp(3C), printf(3S), setjmp(3C), string(3C).

SUPPORT STATUS

Supported.

NAME

cu − call another UNIX system

SYNOPSIS

cu [−sspeed] [−lline] [−h] [−t] [−m] [−o] [−e] [−n]
telno | systemname | dir

DESCRIPTION

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

OPTIONS

Cu accepts the following options and arguments.

−sspeed
: Specifies the transmission speed (110, 150, 300, 600, 1200, 4800, 9600); 300 is the default value. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

−lline
: Specifies a device name to use as the communication line. This can be used to override searching 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 file **/usr/lib/uucp/L-devices**. When the −l and −s options are used simultaneously, cu searches the L-devices file to check if the requested speed for the requested line is available. If so, the connection is made at the requested speed; otherwise an error message is printed and the call is not made. The specified device is generally a directly connected asynchronous line (e.g., **/dev/tty01**). In this case a phone number is not required but the string *dir* may be use to specify a null acu. If the specified device is associated with an auto dialer, a phone number must be provided.

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

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

−m
: Designates a direct line which has modem control.

−e
: Designates that even parity is to be generated for data sent to the remote.

−o
: Designates that odd parity is to be generated for data sent to the remote.

−n
: Requests the phone number to be dialed from the user rather than taking it from the command line.

telno
: When using an automatic dialer the argument is the telephone number with equal signs for secondary dial tone or minus signs for delays, at appropriate places.

systemname
: A *uucp*(1C) system name may be used rather than a phone number; in this case, cu obtains an appropriate direct line or phone number from **/usr/lib/uucp/L.sys** (the appropriate baud

rate is also read with phone numbers). *Cu* tries each phone number or direct line for *systemname* in the L.sys file until a connection is made or all the entries are tried.

dir Using *dir* insures that *cu* uses the line specified by the **−l** option.

PROCESS DESCRIPTION

After making the connection, *cu* runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following:

~. Terminate the conversation.

~! Escape to an interactive shell on the local system.

~!*cmd...*
Run *cmd* on the local system (via **sh −c**).

~$*cmd...*
Run *cmd* locally and send its output to the remote system.

~%**cd**
Change the directory on the local system. NOTE: ~!**cd** causes 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.

~ ~...
Send the line ~... to the remote system.

~%**break**
Transmit a **BREAK** to the remote system.

~%**nostop**
Toggle 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. A line from the remote that begins with ~> initiates an output diversion to a file. The complete sequence is:

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

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

The use of ~%**put** requires *stty*(1) and *cat*(1) on the remote side. It also requires that the current erase and kill characters on the

remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

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

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 ~ ~. For example, *uname*(1) can be executed on Z, X, and Y as follows:

```
uname
Z
~!uname
X
~ ~!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 login to a system connected by a direct line:

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

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

```
cu -s1200 -l /dev/ttyXX dir
```

FILES

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

SEE ALSO

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

DIAGNOSTICS

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

RESTRICTIONS

Cu buffers input internally.
There is an artificial slowing of transmission by *cu* during the ~%put operation so that loss of data is unlikely.
Cu does not support auto-dialing.

SUPPORT STATUS

Supported.

NAME

cut − cut out selected fields of each line of a file

SYNOPSIS

cut −c*list* [file1 file2 ...]
cut −f*list* [−d*char*] [−s] [file1 file2 ...]

DESCRIPTION

Use *cut* to cut out columns from a table or fields from each line of a file; in data base terminology, *cut* implements the projection of a relation.

The fields 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). Either the −c or −f option must be specified.

Cut can be used as a filter; if no files are given, the standard input is used.

A line can have no more than 1023 characters or fields.

No error occurs if a line has fewer fields than a list calls for.

OPTIONS

list Denotes a comma-separated list of integer field numbers in increasing order, with optional − to indicate ranges as in the −o option of *nroff/troff* for page ranges; for example, 1,4,7; 1−3,8; −5,10 (short for 1−5,10); or 3− (short for third through last field).

−c*list* Designates *list* as character positions to be passed For example, −c1−72 would pass the first 72 characters of each line.

−f*list* Designates *list* as a list of fields assumed to be separated in the file by a delimiter character (see −d); for example, −f1,7 copies the first and seventh field only. Lines with no field delimiters are passed through intact (useful for table subheadings), unless −s is specified.

−d*char* Denotes *char* as 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 are passed through untouched.

HINTS

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

EXAMPLES

Map user IDs to names.

```
cut −d; −f1,5 /etc/passwd
```

Set **name** to current login name.

```
name=`who am i | cut −f1 −d" "`
```

DIAGNOSTICS

line too long
A line can have no more than 1023 characters or fields.

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

no fields
The *list* is empty.

SEE ALSO

grep(1), paste(1).

SUPPORT STATUS

Supported.

NAME

cw, checkcw − prepare constant-width text for troff

SYNOPSIS

cw [−lxx] [−rxx] [−fn] [−t] [+t] [−d] [files]

checkcw [−lxx] [−rxx] files

DESCRIPTION

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

Text typeset with the CW font resembles the output of terminals and of line printers. This font is used to typeset examples of programs and of computer output in user manuals, programming texts, etc. It has been designed to be quite distinctive (but not overly obtrusive) when used together with the Times Roman font.

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

The CW font contains the 94 printing ASCII characters:

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

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

Character	*Symbol*	*Troff Name*
Cents sign		\(ct
EBCDIC not sign		\(no
Left arrow	←	\(<-
Right arrow	→	\(->
Down arrow	↓	\(da
Vertical single quote	'	\(fm
Control-shift indicator		\(dg
Visible space indicator		\(sq
Hyphen	-	\(hy

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

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

.CW Start of text to be set in the CW font; .CW causes a break; it can take precisely the same options, in precisely the same format, as are available on the *cw* command line.

.CN End of text to be set in the CW font; .CN causes a break; it can take the same options as are available on the *cw* command line.

.CD Change delimiters and/or settings of other options; takes the same options as are available on the *cw* command line.

.CP *arg1 arg2 arg3 ... argn*
All the arguments (which are delimited like *troff* macro arguments) are concatenated, with the odd-numbered arguments set in the CW font and the even-numbered ones in the prevailing font.

.PC *arg1 arg2 arg3 ... argn*
Same as .CP, except that the even-numbered arguments are set in the CW font and the odd-numbered ones in the prevailing font.

The .CW and .CN requests are meant to bracket text (e.g., a program fragment) that is to be typeset in the CW font as is. Normally, *cw* operates in the *transparent* mode. In that mode, except for the .CD request and the nine special four-character names listed in the table above, every character between .CW and .CN request lines stands for itself. In particular, *cw* arranges for periods (.) and apostrophes (') at the beginning of lines, and backslashes (\) everywhere to be hidden from *troff*. The transparent mode can be turned off (see below), in which case normal *troff* rules apply; in particular, lines that begin with . and ' are passed through untouched (except if they contain delimiters; see below). In either case, *cw* hides the effect of the font changes generated by the .CW and .CN requests; *cw* also defeats all ligatures (fi, ff, etc.) in the CW font.

The only purpose of the .CD request is to allow the changing of various options other than just at the beginning of a document.

The user can also define *delimiters*. The left and right delimiters perform the same function as the .CW/.CN requests; they are meant, however, to enclose CW "words" or "phrases" in running text (see example under *RESTRICTIONS* below). *Cw* treats text between delimiters in the same manner as text enclosed by .CW/.CN pairs, except that, for aesthetic reasons, spaces and backspaces inside .CW/.CN pairs have the same width as other CW characters. While spaces and backspaces between delimiters are half as wide, so they have the same width as spaces in the prevailing text (but are *not* adjustable). Font changes due to delimiters are *not* hidden.

Delimiters have no special meaning inside .CW/.CN pairs.

The options are:

−l*xx*
The one- or two-character string *xx* becomes the left delimiter; if *xx* is omitted, the left delimiter becomes undefined, which it is initially.

−r*xx*
Same for the right delimiter. The left and right delimiters

may (but need not) be different.

−f*n* The CW font is mounted in font position *n*; acceptable values for *n* are 1, 2, and 3 (default is 3, replacing the bold font). This option is only useful at the beginning of a document.

−t Turn transparent mode *off*.

+t Turn transparent mode *on* (this is the initial default).

−d Print current option settings on standard error in the form of *troff* comment lines. This option is meant for debugging.

Cw reads the standard input when no *files* are specified (or when − is specified as the last argument), so it can be used as a filter. Typical usage is: cw *files* | troff ... *Checkcw* checks that left and right delimiters, as well as the .CW/.CN pairs, are properly balanced. It prints out all offending lines.

HINTS

Typical definitions of the .CW and .CN macros meant to be used with the *mm*(5) macro package:

```
.de CW
.DS I
.ps 9
.vs 10.5p
.ta 16m/3u 32m/3u 48m/3u 64m/3u 80m/3u 96m/3u ...
..
.de CN
.ta 0.5i 1i 1.5i 2i 2.5i 3i 3.5i 4i 4.5i 5i 5.5i 6i
.vs
.ps
.DE
..
```

At the very least, the .CW macro should invoke the *troff* no-fill (.nf) mode.

When set in running text, the CW font is meant to be set in the same point size as the rest of the text. In displayed matter, on the other hand, it can often be profitably set one point *smaller* than the prevailing point size. The CW font is sized so that, when it is set in 9-point, there are 12 characters per inch.

Documents that contain CW text may also contain tables and/or equations. If this is the case, the order of preprocessing should be: *cw*, *tbl*, and *eqn*. Usually, the tables contained in such documents do not contain any CW text, although it is entirely possible to have *elements* of the table set in the CW font; of course, care must be taken that *tbl*(1) format information not be modified by *cw*. Attempts to set equations in the CW font are not likely to be either cw pleasing or successful.

In the CW font, overstriking is most easily accomplished with backspaces: letting ← represent a backspace, d←←\(dg yields d Because backspaces are half as wide between delimiters as inside .CW/.CN pairs (-see above), two backspaces are required for each overstrike between delimiters.

FILES

/usr/lib/font/ftCW CW font-width table

SEE ALSO

eqn(1), nroff(1), tbl(1), mm(5), mv(5).

WARNINGS

If text preprocessed by *cw* is to make any sense, it must be set on a typesetter equipped with the CW font or on a STARE facility; on the latter, the CW font appears as bold, but with the proper CW spacing.

RESTRICTIONS

The use of periods (.), backslashes (\), or double quotes (") as delimiters or as arguments to .CP and .PC is greatly discouraged.

Certain CW characters do not concatenate gracefully with certain Times Roman characters, e.g., a CW ampersand (&) followed by a Times Roman comma(,). In such cases, judicious use of *troff* half- and quarter-spaces (\| and \^) is satisfactory, e.g., one should use _&_\^, (rather than just plain _&_,) to obtain &, (assuming that _ is used for both delimiters).

Using *cw* with *nroff* is non-productive.

The output of *cw* is hard to read.

See also *RESTRICTIONS* under *nroff*(1).

SUPPORT STATUS

Supported.

NAME

cxref − generate C program cross-reference

SYNOPSIS

cxref [options] files

DESCRIPTION

Cxref analyzes a collection of C files and attempts to build a cross-reference table. *Cxref* utilizes a special version of *cpp* to include **#define** information in its symbol table. *Cxref* produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the **−c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

OPTIONS

In addition to the **−D**, **−I** and **−U** options (which are identical to their interpretation by *cc*(1)), the following *options* are interpreted by *cxref*:

−c Prints a combined cross-reference of all input files.

−w*<num>*
Formats output no wider than <num> (decimal) columns. <Num> defaults to 80 if <num> is not specified or is less than 51.

−o file Directs output to *file*.

−s Suppresses printing input file names.

−t Formats listing for 80-column width.

FILES

/usr/lib/xcpp special version of C-preprocessor.

SEE ALSO

cc(1).

DIAGNOSTICS

Error messages are unusually cryptic, but usually they mean that the files cannot be compiled.

RESTRICTIONS

Cxref considers a formal argument in a **#define** macro definition to be a declaration of that symbol. For example, a program that includes **#includes ctype.h** contains many declarations of the variable **c**.

SUPPORT STATUS

Supported.

NAME

date − print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

DESCRIPTION

Date outputs or sets the system date and time. If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last two digits of the year number and is optional. The current year is the default if no year is specified. For example:

date 10080045

sets the date to Oct 8, 12:45 AM. The system operates in GMT. *Date* automatically converts to and from local standard and daylight time.

If the argument begins with +, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to *printf*(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and is replaced in the output by its corresponding value. A single % is encoded by %%. *Date* copies all other characters to the output without change. The string is always terminated with a new-line character.

The format field descriptors are:

n	insert a new-line character
t	insert a tab character
m	month of year − 01 to 12
d	day of month − 01 to 31
y	last two digits of year − 00 to 99
D	date as mm/dd/yy
H	hour − 00 to 23
M	minute − 00 to 59
S	second − 00 to 59
T	time as HH:MM:SS
j	day of year − 001 to 366
w	day of week − Sunday = 0
a	abbreviated weekday − Sun to Sat
h	abbreviated month − Jan to Dec
r	time in AM/PM notation

EXAMPLE

date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'

generates as output

DATE: 08/01/76
TIME: 14:45:05

DIAGNOSTICS

No permission	You are not the superuser and try to change the date

bad conversion The date set is syntactically incorrect
bad format character The field descriptor is not recognizable

FILES

/dev/kmem

WARNING

It is a bad practice to change the date while the system is running multi-user.

SUPPORT STATUS

Supported.

NAME

dc − desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. (See bc(1), a preprocessor for *dc* providing infix notation and a C-like syntax which implements functions and 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. All registers are initialized to zero. Registers may also be treated as stacks.

COMMANDS

number

Push the value of the number 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.

+ − / * % ^

Add (+), subtract (−), multiply (*), divide (/), remainder (%), or exponentiate (^) the top two values on the stack. Pop the two entries off the stack; push the result on the stack in their place. Ignore any fractional part of an exponent.

s*x* Pop the top of the stack and store it into a register named *x*, where *x* may be any character.

S*x* Pop the top of the stack and push it on stack *x*.

l*x* Push the value in register *x* on the stack. Register *x* is not altered.

L*x* Pop the top of stack *x* and push it onto the main stack.

d Duplicate the top value on the stack.

p Print the top value on the stack. The top value remains unchanged.

P Interpret the top of the stack as an ASCII string, remove it, and print it.

f Print all values on the stack.

q Exit the program. If executing a string, pop the recursion level by two.

Q Pop the top value on the stack and then pop the string execution level by that value.

x Treat the top element of the stack as a character string and execute it as a string of *dc* commands.

X Replace the number on the top of the stack with its scale factor.

[...] Push the bracketed ASCII string onto the top of the stack.

<*x* >*x* =*x*
Pop the top two elements of the stack and compare them. Evaluate register *x* if the elements obey the stated relation.

v Replace the top element on the stack by its square root. Take into account any existing fractional part of the argument, but otherwise ignore the scale factor.

! Interpret the rest of the line as a UNIX System command.

c Pop all values on the stack.

i Pop the top value on the stack and use it as the number radix for further input.

I Push the input base on the top of the stack.

o Pop the top value on the stack and use it as the number radix for further output.

O Push the output base on the top of the stack.

k Pop the top of the stack, and use that value 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 is reasonable if all are changed together.

z Push the stack level onto the stack.

Z Replace the number on the top of the stack with its length.

? Take a line of input from the input source (usually the terminal) and execute it.

; : Denote array operations.

EXAMPLE

This example prints the first ten values of n!:

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

SEE ALSO

dc(1), *Arbitrary Precision Desk Calculator Language (BC)* in the Support Tools Guide.

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.

SUPPORT STATUS
Supported.

NAME

dd − convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies the input file specified using **if** to the output file specified by **of** with any conversion(s) specified by **conv**. The input and output size may be specified to take advantage of raw physical i/o.

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968.

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

OPTIONS

if=*ifile* Denotes *ifile* as the input file; if not specified, standard input is assumed.

of=*ofile* Denotes *ofile* as the output file; if not specified, standard output is assumed.

ibs=*n* Sets input block size to *n* bytes, where *n* is a number which may be followed by **k**, **b** or **w** to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers, *nxm* may be used for *n* to indicate a product. If not specified, 512 bytes is assumed.

obs=*n* Sets output block size to *n* bytes (see *ibs*). If not specified, 512 bytes is assumed.

bs=*n* Sets both input and output block size to *n* bytes (see *ibs*), overriding **obs** and **ibs**. Also, if no conversion is specified, the *bs* option is particularly efficient since no in-core copying need be done. If not specified, 512 bytes is assumed.

cbs=*n* Sets conversion buffer size to *n* bytes (see *ibs*); used only if *ascii* or *ebcdic* conversion is specified. If not specified, 512 bytes is assumed.

skip=*n* Skips *n* blocks from the beginning of the input file before copying.

seek=*n* Skips *n* blocks from the beginning of the output file before copying.

count=*n* Copies only *n* input blocks.

conv=*ascii* Converts EBCDIC to ASCII. *Dd* places *cbs* characters into the conversion buffer, converts them to ASCII, trims trailing blanks, and adds a new-line before copying the line to the output file.

ebcdic Converts ASCII to EBCDIC. *Dd* reads ASCII characters into the conversion buffer from the input block; *dd* then converts the characters to EBCDIC and adds blanks to make an output block of sized *cbs*.

ibm Maps ASCII to EBCDIC slightly differently than *ebcdic* does. *Ibm* corresponds better to certain IBM print train conventions.

lcase Maps alphabetics to lower case.

ucase Maps alphabetics to upper case.

swab Swaps every pair of bytes.

noerror Indicates to continue processing on an error.

sync Pads every input block with blanks to *ibs* characters.

...,... Denotes several comma-separated conversions.

EXAMPLE

This command reads a streaming tape blocked 8192-bytes per block into file x.

```
dd if=/dev/rstp/0yy of=x bs=8192
```

Dd is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary block sizes.

SEE ALSO

cp(1).

DIAGNOSTICS

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

RESTRICTIONS

New-lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC.

SUPPORT STATUS

Supported.

NAME

delta − make a delta (change) to an SCCS file

SYNOPSIS

delta [−rSID] [−s] [−n] [−glist] [−m[mrlist]] [−y[comment]] [−p] files

DESCRIPTION

Delta permanently changes the named SCCS file according to the changes in the *g-file* generated by a previous *get*(1). If a directory is named, *delta* behaves as though each file in the directory were specified as a named file; non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a filename of − is given, the standard input is read (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

If the standard input (−) is specified on the *delta* command line, the **−m** (if necessary) and **−y** options *must* also be present. Omission of these options causes an error to occur.

Delta may issue prompts on the standard output depending upon the options specified and flags (see *admin*(1)) that may be present in the SCCS file (see **−m** and **−y** below).

The edited *g-file* is normally removed when delta processing is completed (see **−n**).

If the SCCS file has the **v** flag set (see *admin*(1)) then a Modification Request (MR) number *must* be supplied as the reason for creating the new delta. (see **−m**).

If the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which validates the correctness of the MR numbers. If a non-zero exit status is returned from MR number validation program, *delta* terminates (it is assumed that the MR numbers were not all valid).

OPTIONS

Options apply independently to each named file.

−r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this option is necessary only if two or more outstanding *get*s for editing (get −e) on the same SCCS file were done by the same person (login name). The *SID* value specified can be either the *SID* specified on the *get* command line or the *SID* to be made as reported by the *get* command (see *get*(1)). Delta prints a diagnostic message if the specified *SID* is ambiguous, or, if the *SID* was necessary and was omitted on the command line.

−s Suppresses the printing of the *SID* of the created delta, as well as the number of lines inserted, deleted and unchanged in the SCCS file. These are normally printed on the standard output.

−n Retains the edited *g-file* which is normally removed at completion of delta processing.

−g*list* Specifies a *list* (see *get*(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (*SID*) created by this delta.

−m[*mrlist*] Specifies the Modification Request numbers in *mrlist* which are the reasons for creating the new delta.

If −m is not used and the standard input is a terminal, delta prompts **MRs?** on the standard output before reading the standard input; if the standard input is not a terminal, delta does not prompt. The **MRs?** prompt always precedes the **comments?** prompt (see −y).

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

−y[*comment*]

Denotes *comment* as the reason for making the delta. A null string is considered a valid *comment*.

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

−p Prints (on the standard output) the SCCS file differences in a *diff*(1) format before and after the delta is applied.

FILES

All files of the form *?*-file are explained in the *Source Code Control System User Guide* in the Support Tools Guide. The naming convention for these files is also described there.

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the file retrieved by *get* and the *g-file*.

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see *sccsfile*(5)) and causes an

error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get/delta* sequences should be used.

RESTRICTIONS

Comments are limited to text strings of at most 512 characters.

SEE ALSO

admin(1), bdiff(1), cdc(1), get(1), help(1), prs(1), rmdel(1), sccsfile(4). *Source Code Control System User Guide* in the Support Tools Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

SUPPORT STATUS

Supported.

NAME

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

SYNOPSIS

deroff [−mx] [−w] [files]

DESCRIPTION

Deroff reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between .EQ and .EN lines, and between delimiters), and *tbl*(1) descriptions, replacing them with white space (blanks and blank lines) when appropriate, 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.

OPTIONS

-m*x* Denotes special handling of macros and may be followed by an **m** or **l**.

The **−mm** option interprets the macros so that only running text is output (i.e., no text from macro lines.)

The **−ml** option invokes the **−mm** option and also deletes lists associated with the **mm** macros.

−w Outputs each input line as a list of *words*, one *word* per line. The output follows the original, with all deletions mentioned above. In text, a *word* is any string that *contains* at least two letters and is composed of letters, digits, ampersands (&), and apostrophes (′); in a macro call, however, a *word* is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from *words*.

SEE ALSO

eqn(1), nroff(1), tbl(1), troff(1).

RESTRICTIONS

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.

SUPPORT STATUS

Supported.

NAME

diff − differential file comparator

SYNOPSIS

diff [**−efbh**] file1 file2

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (or *file2*) is −, the standard input is used. If *file1* (or *file2*) is a directory, then a file in that directory with the name *file2* (or *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*(1) 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 >.

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

OPTIONS

−b Ignore trailing blanks (spaces and tabs) and compare other strings of blanks equal.

−e Produce a script of **a**, **c**, and **d** commands for the editor *ed*, which will recreate *file2* from *file1*.

−f Produce a script similar to **−e**, not useful with *ed*, in the opposite order.

−h Produce the differences faster. The **−h** option works only when changed stretches are short and well separated. It does work on files of unlimited length. Options **−e** and **−f** cannot be used with **−h**.

HINTS

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
```

EXAMPLE

To compare the files *ch2.old* and *ch2* and store the differences in the file *ch2.diff*, enter

```
diff ch2.old ch2 > ch2.diff
```

To compare the file */etc/passwd* to the file *passwd* in the current directory, enter

diff passwd /etc

FILES

/tmp/d?????
/usr/lib/diffh for **−h**

SEE ALSO

cmp(1), comm(1), ed(1).

DIAGNOSTICS

The exit status is 0 for no differences, 1 for some differences, or 2 for major differences, possibly errors.

Missing newline at end of file X is output if the last line of file X does not have a newline. If the lines are different, they are flagged and output although the output seems to indicate they are the same.

RESTRICTIONS

Editing scripts produced under the **−e** or **−f** option may arbitrarily display a line with a single period on the output.

SUPPORT STATUS

Supported.

NAME

diff3 − 3-way differential file comparison

SYNOPSIS

diff3 [−ex3] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	*file1* is different
====2	*file2* is different
====3	*file3* is different

The type of change made 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.

OPTIONS

−e publishes a script for the editor *ed* that incorporates into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged ==== and ====3.

−x produces a script to incorporate only changes flagged ====.

−3 produces a script to incorporate only changes flagged ====3.

HINTS

The following command applies the script (resulting from the use of any options) to *file1*.

(cat script; echo '1,$p') | ed − file1

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO

diff(1).

RESTRICTIONS

Ed(1) uses a single period to terminate appends, changes, etc; therefore, a text line that consists of a single . confuses *ed*(1), limiting the usefulness of −e.

Files longer than 64K bytes do not work.

SUPPORT STATUS

Supported.

NAME

diffmk − mark differences between files

SYNOPSIS

diffmk name1 name2 name3

DESCRIPTION

Diffmk compares two versions of a file and creates a third file that includes *change mark* commands for *nroff* or *troff*(1). *Name1* and *name2* are the old and new versions of the file. *Diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter *change mark* (.mc) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

If the characters | and * are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shell procedure).

EXAMPLE

Diffmk can be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file **macs** contains:

```
.pl 1
.ll 77
.nf
.eo
.nc \
```

The .ll request might specify a different line length, depending on the nature of the program being printed. The **.eo** and **.nc** requests are probably needed only for C programs.

SEE ALSO

diff(1), nroff(1), troff(1).

RESTRICTIONS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing **.sp** by **.sp 2** produces a *change mark* on the preceding or following line of output.

SUPPORT STATUS

Supported.

NAME

dircmp − directory comparison

SYNOPSIS

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

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the filenames common to both directories have the same content.

OPTIONS

−d Compare the content 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).

SUPPORT STATUS

Supported.

NAME

dis − disassembler

SYNOPSIS

dis [−o] [−V] [−L] [−d sec] [−da sec] [−F function] [−t sec] [−l string] files

DESCRIPTION

The *dis* command produces an assembly language listing of each of its object *file* arguments. The listing includes assembly statements and the binary that produced those statements.

On output, a number enclosed in brackets at the beginning of a line, such as [5], represents that the C breakpointable line number starts with the following instruction. An expression such as <40> in the operand field, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A C function name appears in the first column, followed by ().

Release 2.x archive format is supported permitting the disassembly of archive modules from Release 2.x libraries transparently.

OPTIONS

The following options are interpreted by the disassembler and may be specified in any order.

−o	Print numbers in octal. Default is hexadecimal.
−V	Write the version number of the disassembler to standard error.
−L	Invoke a lookup of C source labels in the symbol table for subsequent printing.
−d *sec*	Disassemble the named section as data, printing the offset of the data from the beginning of the section.
−da *sec*	Disassemble the named section as data, printing the actual address of the data.
−t *sec*	Disassemble the named section as text.
−l *string*	Disassemble the library file specified as *string*. For example, one would issue the command **dis −l x −l z** to disassemble **libx.a** and **libz.a**. All libraries are assumed to be in **/lib**.

If the **−d**, **−da**, or **−t** options are specified, only those named sections from each user supplied filename are disassembled. Otherwise, all sections containing text are disassembled.

If the **−F** option is specified, only those named functions from each user supplied filename are disassembled.

SEE ALSO

as(1), cc(1), ld(1).

DIAGNOSTICS

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

SUPPORT STATUS
Supported.

NAME

du − summarize disk usage

SYNOPSIS

du [**−ars**] [names]

DESCRIPTION

Du prints the number of blocks contained in all files and directories within each directory and file specified by the *names* argument. The number of blocks includes the indirect blocks of the file. If *names* is missing, . is used. If no options are given, *du* generates an entry for each directory only, and generates no messages about directories that cannot be read, files that cannot be opened, etc.

A file with more than one link is only counted once.

OPTIONS

−s Prints only the grand total for each of the specified names.

−a Generates and prints an entry for each file.

−r Prints messages about directories that cannot be read, files that cannot opened, etc.

RESTRICTIONS

If the **−a** option is not used, non-directories given as arguments are not listed.

If there are too many distinct linked files, *du* will count the excess files more than once.

Files with holes in them will get an incorrect block count.

SUPPORT STATUS

Supported.

NAME

dump − dump selected parts of an object file

SYNOPSIS

dump [**−acfghlorst**] [**−z** name] files

DESCRIPTION

The *dump* command dumps selected parts of each of its object *file* arguments.

The *dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hexadecimal, octal or decimal representation as appropriate.

This command accepts both object files and archives of object files. It processes each file argument according to one or more of the options below.

OPTIONS

−a Dump the archive header of each member of each archive file argument.

−c Dump the string table.

−f Dump each file header.

−g Dump the global symbols in the symbol table of an archive.

−h Dump section headers.

−l Dump line number information.

−o Dump each optional header.

−r Dump relocation information.

−s Dump section contents.

−t Dump symbol table entries.

−z name
Dump line number entries for the named function.

The following *modifiers* are used in conjunction with the options listed above to modify their capabilities. Blanks separating an *option* and its *modifier* are optional.

−d number Dump the section number or range of sections starting at *number* and ending either at the last section number or *number* specified by **+d**.

+d number Dump sections in the range either beginning with first section or beginning with section specified by **−d**.

−n name Dump information pertaining only to the named entity. This *modifier* applies to **−h**, **−s**, **−r**, **−l**, and **−t**.

−p Suppress printing of headers.

−t index Dump only the indexed symbol table entry. The **−t** used in conjunction with **+t**, specifies a range of symbol table entries.

+t index Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first

symbol table entry or at the entry specified by the −t option.

−u Underline the name of the file for emphasis.

−v Dump information in symbolic representation rather than numeric (e.g., C_STATIC instead of 0X02). This *modifier* can be used with all the above options except −s and −o options of *dump*.

−z name,number
Dump line number entry or range of line numbers starting at *number* for the named function.

+z **number** Dump line numbers starting at either function *name* or *number* specified by −z up to *number* specified by +z.

NOTE

Archives from Release 2.x and any later releases may be dumped permitting transparent use of *dump* with Release 2.x archive libraries. Symbol table mapping done by *ld*(1) and *nm*(1) are not done.,

SEE ALSO

ld(1), nm(1), a.out(4), ar(4).

SUPPORT STATUS

Supported.

NAME

echo – echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
****	backslash
*****n*	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number *n*, which must start with a zero.

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

SEE ALSO

sh(1).

SUPPORT STATUS

Supported.

NAME

ed, red − text editor

SYNOPSIS

ed [−] [**−p** string] [**−x**] [file]

red [−] [**−p** string] [**−x**] [file]

DESCRIPTION

Ed is the standard text editor. If argument *file* is given, *ed* reads *file* into the edit buffer so that the file can be edited. The option − suppresses the character counts returned by the *e*, *r*, and *w* commands; it also supresses diagnostics from the *e* and *q* commands, and the ! prompt after a *!shell command*. The **−p** option allows the user to specify a prompt string. If the **−x** option is used, an *x* command is simulated first to handle an encrypted file.

Ed operates on a copy of the text being edited, which resides in a temporary file called the edit *buffer*. There is only one buffer. Changes made in the buffer have no effect on the file until a *w* (write) command is given.

Red is a restricted version of *ed*. It only allows editing of files in the current directory. It prohibits executing shell commands via *!shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty −tabs** or **stty tab3** mode (see *stty*(1)), the specified tab stops are automatically used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10 and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, *ed* expands typed tab characters to every eighth column, as is normal.

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 into the buffer. While *ed* is accepting text, it is in *input mode*. In *input mode*, *ed* does not recognize commands; *ed* merely collects input. To exit input mode, type a period (.) alone at the beginning of a line.

REGULAR EXPRESSIONS (REs)

Ed supports a limited form of *regular expression* notation; *ed* uses regular expressions in addresses to specify lines, and in some commands (such as *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 *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character RE*s match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]; see 1.4 below).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

c. $ (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string.

If 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 a string of characters.

The following rules may be used to construct *RE*s from one-character REs:

2.1 A one-character RE matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) 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*\} 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 matches the concatenation of the strings matched by each component that was concatenated.

2.5 A RE enclosed between the character sequences \(and \) 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.

An *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2 A currency symbol ($) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction ^ *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 *HINTS*.

ADDRESSES

To understand addressing in *ed* it is necessary to know that at all times there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character . addresses the current line.
2. The character $ addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. *'x* addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that *ed* searches the entire buffer. See also *HINTS*.

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 *HINTS*.

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 *ed* increments (+) or decrements (-) the address by 1. 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, a comma separates addresses from each other. 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.

COMMANDS

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 invalid 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.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a ? and returns to *its* command level.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; . is the last appended line, or, if no lines were appended, the addressed line. Address 0 is valid for this command: *ed* places the appended text at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line, including the newline character.

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is the last line input, or, if no lines were input, the first line that was not deleted.

(.,.)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 deletes the entire contents of the buffer, and then reads in the named file; . becomes 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 printed; *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 assumed to be a shell (*sh*(1)) command whose output will be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

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, *g* first marks every line that matches the given RE. Then, for every such line, *g* executes the given *command list* 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 have a \ before the newline. The . terminating input mode may be omitted if it is the last line of the *command list*. An empty *command list* is equivalent to the *p* command.

The *a*, *i*, and *c* commands and associated input are valid commands in the command list. The *g*, *G*, *v*, and *V* commands are *not* valid commands in the *command list*. See also *RESTRICTIONS* and *HINTS*.

(1,$)G/*RE*/

In the interactive *G*lobal command, *G* first marks every line that matches the given RE. Then, for every such line, *G* prints that line, and . changes to that line. Then *G* accepts any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) from the terminal and executes that command. The next marked line is then printed, and the process repeats.

A new-line acts as a null command; an ampersand (&) re-executes 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 prints a short error message that explains the reason for the most recent ? diagnostic.

H

The *H*elp command puts *ed* in a mode in which error messages are printed for all subsequent ? diagnostics. *H* also explains 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; . becomes the last inserted line, or, if no lines were inserted, the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line including the newline character.

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command has no effect.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. Subsequently, the address *'x* 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 mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are split so that all of the line may be viewed. An *l* command

may be appended to any other 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 valid for *a*; *ed* moves the addressed line(s) to the beginning of the file; *m* generates an error if address *a* falls within the range of moved lines; . becomes the last line moved.

(.,.)n

The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; . becomes the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p

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

P

The editor prompts 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 exits *ed*. No automatic write of a file is done (see *DIAGNOSTICS* below).

Q

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

($)r *file*

The *r*ead command reads in the named file after the addressed line. If no file name is specified, 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 valid for *r*; *ed* reads the file in at the beginning of the buffer. If the read is successful, *ed* prints the number of characters read; . is set to the last line read in. If *file* is replaced by !, the rest of the line is assumed to be a shell (*sh*(1)) command whose output will 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*/
(.,.)s/*RE*/*replacement*/g
(.,.)s/*RE*/*replacement*/n

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* string if the global replacement indicator g

appears after the command. If a number n appears after the command, the n th occurrence of the matched string on each addressed line is replaced. If neither the global indicator nor the number indicator appears, *ed* only replaces the first occurence of the matched string. If the substitution fails on *all* addressed lines, an error message appears. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . becomes the last line on which a substitution occurred. See also *HINTS*.

Ed replaces an ampersand (&) appearing in the *replacement* with the string matching the RE on the current line. Preceding & with a \ suppresses the special meaning of & in this context.

Ed replaces the characters *n*, where *n* is a digit, with 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*, *ed* uses the *replacement* used in the most recent substitute command 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 *ed* places a *copy* of the addressed lines after address *a* (which may be 0); . becomes 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 *ed* executes the *command list* 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, *ed* creates it with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. *Ed* does not change the currently-remembered file name 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, *ed* prints the number of characters written. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

X

A key string is demanded from the standard input. Subsequent *e*, *r*, and *w* commands encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

(**$**)=

The line number of the addressed line is typed; . is unchanged by this command.

!*shell command*

The remainder of the line after the ! is sent to the UNIX 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, !! repeats 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 prints the addressed line. A new-line alone is equivalent to .+1p; it is useful for stepping forward through the buffer. The current line (.) becomes the addressed line.

HINTS

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 *ed* prints the addressed line. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

FILES

/tmp/e#	temporary; # is the process number.
ed.hup	*ed* saves work here if the terminal is hung up.

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 the edit buffer via the *e* or *q* commands: *ed* prints ? and allows one to continue editing. A second *e* or *q*

command at this point destroys the buffer. The — command-line option inhibits this feature.

SEE ALSO

crypt(1), grep(1), sed(1), sh(1), stty(1), fspec(4), regexp(5).
Text Editor (ed) — Basic and *Text Editor (ed) — Advanced* in the Editing Guide.

RESTRICTIONS

Some size limitations:

512 characters per line

256 characters per global command list

64 characters per file name

128K characters in the edit buffer

The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

A *!* command cannot be subject to a *g* or a *v* command.

The *!* command and the ! escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).

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

The *l* command mishandles DEL.

Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.

Characters are masked to 7 bits on input.

If the editor input is from a command file (for example, ed file < ed-cmd-file), the editor exits at the first failure of a command in the command file.

SUPPORT STATUS

Supported. The *crypt* feature is available only in the United States.

NAME

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

SYNOPSIS

edit [**−r**] 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. The following brief introduction should help you get started with *edit.*

Basic edit commands

edit

To edit the contents of a file, enter the shell command

edit *fname*

where *fname* is the name of an existing file or a file your wish to create. (To enter a command, type the command then press return or NEWLINE.) *Edit* copies the file *fname* into the edit buffer and prints a file size message indicating how many lines and characters are in the edit buffer. If *fname* is a new file, *edit* prints *fname* followed by [**Newfile**]. After printing the new file or file size message, *edit* prompts for a command with a colon.

current line, print, and delete

Most *edit* commands affect the *current line* in the buffer if you do not specify another line or lines in the command. For example, if you enter

print

or its abbreviated form **p**, *edit* prints the current line. To move the current line down a line, press NEWLINE; *edit* prints the new current line.

To delete the current line, enter **d**. *Edit* deletes the line and prints the new current line. When *edit* is first entered, the current line is the last line of the file. If you **delete** this last line, the the new last line of the file becomes the current line. In all other cases, the line *after* the deleted line becomes the current line.

Edit numbers the lines in the buffer; the first line has number 1. If you enter **1**, *edit* prints the first line. Enter **d** and *edit* deletes the first line, line 2 becomes line 1, and *edit* prints the current line (the new line 1).

append and insert

To add lines to a newly created file, or to an existing file, use the **append** (**a**) command. Enter **a**, and *edit* reads subsequent lines from your terminal into the file, adding these new lines after the current line. To signal the end of the input lines, enter a period on a line by itself. The last line entered before the period becomes the new current line.

The **insert** (**i**) works just like **append** but *edit* places the input lines *before* the current line, rather than after it.

substitute

To change some text within the current line, use the **substitute** (**s**)

command. Enter s/*oldtext*/*newtext*/ where *oldtext* is the old characters you want replace with the new characters, *newtext*.

To delete some characters, use the *substitute* command as follows: s/*oldtext*// where *oldtext* is the text your want to delete.

file, write, and quit

The command **file** (**f**) prints how many lines there are in the edit buffer and prints *modified* if the buffer has been changed.

To replace the file with the buffer text (i.e. to save the changes) enter a **write** (**w**) command. You may then leave the editor using the **quit** (**q**) command. If you *edit* a file, but make no changes to it, it is not necessary (but does no harm) to write the file.

If you try to **quit** *edit* after modifying the buffer without writing the file, *edit* warns that there has been *No write since last change*. If you do not wish to save the changes in the edit buffer, enter the **quit!** or **q!** command. *Edit* discards the buffer (irretrievably) and you return to the shell.

Other commands

The above basic commands are adequate for making any changes you need to make. A few other commands are useful to know, however, if you intend to use edit more than a few times.

change and line ranges

The **change** (**c**) command changes the current line to a sequence of lines you supply (as in **append** you enter a sequence of lines ending with a line consisting of only a period).

Change changes more than one line if you specify the line numbers of the lines you want to **change**. For example, **3,5change** changes lines 3 through 5 to the lines you supply. The line range also works in other commands; **1,20p** prints the first 20 lines of the file.

undo

The **undo** (**u**) command reverses the effect of the last command which changed the contents of the buffer (not necessarily the current line). For example, if a **substitute** command does not do what you intended, enter **undo**; *edit* restores the old contents of the line. You can also **undo** an **undo** command if you continue to change your mind.

Edit prints a message when a command (including *undo*) affects more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an *undo* and looking to see what happened. If you decide that the change is ok, then you can *undo* again to get it back.

Note that commands such as *write* and *quit* cannot be undone.

^D and z (viewing lines) and count

There are two ways to look at several lines in the buffer: **^D** and **z**.

^D, executed by holding down the *control* key and and pressing the *D* key (the *control* key is similar to the *shift* key) prints the next half screen of lines on a CRT or the next 12 lines on a hard-copy terminal.

The z command is a more versatile way to view several lines. z. clears the screen and prints a full screen of lines with the current line near the center of the screen. The last line printed becomes the new current line. z− prints the screenful of lines which precede the current line, printing the current line last. z+ prints the screenful of lines which follow the current line.

To view less that a screenful of lines, append the number of lines you wish to view to z− or z+. For example, z+5 prints the next 5 lines; 5 is the *count* of lines.

Count works for other commands, too; delete 5, for example, deletes 5 lines beginning with the current line.

/ and ? (search)

To locate strings in the file you can use line numbers if you know them; but the line numbers change when you insert and delete lines, so the correct line number may be difficult to remember. It may be easier to search for the strings using / or ?.

/text/ searches forward for *text*; *?text?* searches backward for *text*. If either search encounters the end of the file in its direction without finding *text*, the search wraps around to the other end of the file and continues to search back to the current line. The line where *text* is found becomes the new current line.

A useful feature 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.

special line numbers

The current line is symbolized by the period, and the last line of the file is symbolized by the $ symbol. This shorthand is useful in expressing line ranges in commands, such as .,$**print**, which prints the lines from the current line through the last line. Arithmetic line references are also possible: line $-5 is the fifth line before the last line, and line ".+20" is the line 20 lines after the current line.

copying and moving text

There are two way to move and copy text: with buffers, and without buffers. Copying or moving text from one file to another requires the use of buffers; moving text around within a file does not. Both ways, however, require that you know the line numbers of the first and last lines of the texts you are going to move.

To move some text from one place in a file to another, you can use the **move** command. For example, 1,5**move**$ moves lines 1 through 5 to the end of the file. To copy text within a file, you can use the **copy** command. 1,5**copy**10 puts a copy of the first five lines of the file after line 10. These commands are straightforward, but they have a drawback: you must know exactly where the lines are going before you can move or copy them, and if you move them to the wrong place, you must use the *undo* command and do the command over again, or figure out the new line numbers.

Buffering is a more versatile way of moving text. *Edit* has 26 buffers named a through z. Text put into these buffers can be

copied out of the buffers any time before you exit *edit;* you do not have to know the destination of the text when you put it into the buffer.

To copy text into the buffer, use the *yank* command. **1,5yank t** copies lines 1 through 5 into buffer t.

To move text into a buffer, deleting the lines from the file, use the **delete** command followed by the buffer name. **1,5d z** moves lines 1 through 5 into buffer z.

To copy the buffered text back into the file, use the **put** command. **Put z**, for example, copies the contents of buffer z after the current line. To copy the buffered text into a second file, you must **edit** that file. If you made any changes to the current file that you want to save, enter **write** before using **edit**. If you do not wish to save the changes, use **edit!**. In either case, enter **edit** *fname* (or **edit!** *fname*) where *fname* is the name of the file where the buffered text is to be copied. Once in the other file, **put** the text as usual.

Note that the buffer contents change only when a new **yank** or **delete** directs lines to the buffer. When **edit** is exited (i.e. a **quit** is issued) the buffer contents are lost.

SEE ALSO

ex(1), vi(1).
Text Editor (ex) in the Editing Guide.

RESTRICTIONS

See *ex*(1).

SUPPORT STATUS

Supported.

NAME

enable, disable − enable/disable LP printers

SYNOPSIS

enable printers
disable [−c] [−r[reason]] printers

DESCRIPTION

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

Disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). Any requests that are currently printing on the designated printers are reprinted in their entirety either on the same printer or on another member of the same class unless −c is specified. Use *lpstat*(1) to find the status of printers.

OPTIONS

−c — Cancels 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 is used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).

SUPPORT STATUS

Not supported.

NAME

env − set environment for command execution

SYNOPSIS

env [−] [name=value] ... [command args]

DESCRIPTION

Env obtains the current *environment*, modifies it according to the *env* 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).

SUPPORT STATUS

Supported.

NAME

eqn, neqn, checkeq − format mathematical text for nroff or troff

SYNOPSIS

eqn [−dxy] [−pn] [−sn] [−fn] [−Tdest] [files]

neqn [−dxy] [−pn] [−sn] [−fn] [files]

checkeq [files]

DESCRIPTION

Eqn is a *troff*(1) preprocessor for typesetting mathematical text on a phototypesetter; *neqn* is used for the same purpose with *nroff*(1) on typewriter-like terminals. Usage is almost always:

```
eqn files | troff
neqn files | nroff
```

If no files are specified, or if − is specified as the last argument, *eqn* and *neqn* read the standard input.

A line beginning with .EQ marks the start of an equation; the end of an equation is marked by a line beginning with .EN. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as *delimiters*; subsequent text between delimiters is then treated as *eqn* input. Delimiters may be set to characters *x* and *y* with **delim** *xy* between .EQ and .EN (see also option −**d**). The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by **delim off**. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

Checkeq reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within *eqn* are separated by spaces, tabs, new-lines, braces, double quotes, tildes, and circumflexes. Braces { } are used for grouping; generally speaking, anywhere a single character such as *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub j* makes x_j, *a sub k sup 2* produces a_k^2, while e^{x+y} is made with *e sup {x sup 2 + y sup 2}*.

Fractions are made with **over**: *a over b* yields $\frac{a}{b}$; **sqrt** makes square roots: *1 over sqrt {ax sup 2+bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits: $\lim_{n \to \infty} \sum_{0}^{n} x_i$ is made with *lim from {n −> inf } sum from 0 to n x sub i*.

Left and right brackets, braces, etc., of the right height are made with **left** and **right**:

left [x sup 2 + y sup 2 over alpha right] = 1 produces $\left[x^2+\frac{y^2}{\alpha}\right] = 1$. Valid characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket). A **left** bracketing character need not have a matching **right** bracketing character.

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. Piles may have arbitrary numbers of elements; **lpile** left-justifies, **pile** and **cpile** center (but with different vertical spacing), and **rpile** right justifies.

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x} = \bar{f}(t)$, y dotdot bar = n under is $\ddot{\bar{y}} = \underline{n}$, and *x vec = y dyad* is $\vec{x} = \vec{\vec{y}}$.

Point sizes and fonts can be changed with **size** *n* or **size** $\pm n$, **roman**, **italic**, **bold**, and **font** *n*. Point sizes and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, (see also options **−s** and **−f**).

Normally, subscripts and superscripts are reduced by 3 points from the previous size (see option **−p**).

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

define token % text %

defines a token called *token* that will be replaced by *text* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords such as **sum** ($\sum$), **int** ($\int$), **inf** (∞), and shorthands such as >= ($\geq$), != ($\neq$), and −> ($\rightarrow$) are recognized. Greek letters are spelled out in the desired case, as in **alpha** (α), or **GAMMA** (Γ). Mathematical words such as **sin**, **cos**, and **log** are made Roman automatically. *Troff*(1) four-character escapes such as **\(dd** (‡) and **\(co** (©) may be used anywhere. Strings enclosed in double quotes are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff*(1) when all else fails. Full details are given in the manual cited below.

OPTIONS

−d*xy*

Set *x* as the left delimiter and *y* as the right delimiter for *eqn*

text.

−s*n* Set *n* to be the point size for the document.

−f*n* Set *n* to be the font number for the document.

−p*n* Reduce point size by *n* for the subscripts and superscripts.

−Tdest
Prepare output for the typesetter *dest*.option. Supported devices are −Taps (Autologic APS-5), −TX97 (Xerox 9700), −Ti10 (Imagen Imprint-10), and −Tcat (Wang CAT). Default is −Taps.

SEE ALSO

cw(1), mm(1), mmt(1), nroff(1), tbl(1), eqnchar(5), mm(5), mv(5).

RESTRICTIONS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold "12.3"**.
See also *RESTRICTIONS* under *troff*(1).

SUPPORT STATUS

Supported.

NAME

error − analyze and disperse compiler error messages

SYNOPSIS

error [**−n**] [**−s**] [**−q**] [**−v**] [**−t** suffixlist] [**−I** ignore-file] [name]

DESCRIPTION

Error analyzes and optionally disperses the diagnostic error messages produced by a number of compilers and language processors to the source file and line where the errors occurred. It can replace the traditional methods of writing abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously without machinations of multiple windows in a screen editor.

Error looks at the error messages, either from the specified file *name* or from the standard input, and attempts to determine which language processor produced each error message, determines the source file and line number to which the error message refers, determines if the error message is to be ignored or not, and inserts the (possibly slightly modified) error message into the source file as a comment on the line preceding the line to which the error message refers. Error messages which cannot be categorized by language processor or content are not inserted into any file, but are sent to the standard output. *Error* touches source files only after all input has been read. By specifying the **−q** query option, the user is asked to confirm any potentially dangerous (such as touching a file) or verbose action. If the **−t** touch option and associated suffix list is given, *error* restricts itself to touch only those files with a suffix in the suffix list. *Error* also can be asked (by specifying **−v**) to invoke *vi*(1) on the files in which error messages were inserted; this makes unnecessary the need to remember the names of the files with errors.

Error is intended to be run with its standard input connected via a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Therefore, both error sources should be piped together into *error*. For example, when using the *csh*(1) syntax,

```
make −s lint |& error −q −v
```

analyzes all the error messages produced by whatever programs *make*(1) runs when making lint.

Error knows about the error messages produced by: *make*, *cc*, *cpp*, *ccom*, *as*, *ld*, *lint*, and *f77*. *Error* knows a standard format for error messages produced by the language processors, so is sensitive to changes in these formats. For all languages except Pascal, error messages are restricted to be on one line. Some error messages refer to more than one line in more than one file; *error* duplicates the error message and inserts it at all of the places referenced.

Error does one of six things with error messages.

synchronize
: Some language processors produce short errors describing which file it is processing. *Error* uses these to determine the file name for languages that do not include the file name in each error message. These synchronization messages are consumed entirely by *error*.

discard
: Error messages from *lint* that refer to one of the two *lint* libraries, **/usr/lib/llib-lc** and **/usr/lib/llib-port** are discarded to prevent accidently touching these libraries. These error messages are consumed entirely by *error*.

nullify
: Error messages from *lint* can be nullified if they refer to a specific function which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named **.errorrc** in the users home directory or from the file named by the −I option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

not file specific
: Error messages that cannot be intuited are grouped together and written to the standard output before any files are touched. They are not inserted into any source file.

file specific
: Error messages that refer to a specific file, but to no specific line, are written to the standard output when that file is touched.

true errors
: Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are candidates for inserting into the file to which they refer. Other error messages are consumed entirely by *error* or are written to the standard output. *Error* inserts the error messages into the source file on the line preceding the line the language processor found in error. Each error message is turned into a one line comment for the language, and is internally flagged with the string "###" at the beginning of the error and "%%%" at the end of the error. This makes pattern searching for errors easier with an editor and allows the messages to be easily removed. In addition, each error message contains the source line number for the line to which the message refers. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can cause problems with a future compilation. To avoid this, programs with comments and source

on the same line should be formatted so that language statements appear before comments.

Error catches interrupt and terminate signals, and if in the insertion phase, orderly terminates.

OPTIONS

Options available with *error* are:

−n Do *not* touch any files; all error messages are sent to the standard output.

−q Query whether the file should be touched. A "y" or "n" response to the question is necessary to continue. Absence of the **−q** option implies that all referenced files (except those referring to discarded error messages) are to be touched.

−v After all files are touched, overlay the visual editor *vi* with it set up to edit all files touched, and positioned in the first touched file at the first error. If *vi*(1) cannot be found, try *ex*(1) or *ed*(1) from standard places.

−t Take the following argument as a suffix list. Files whose suffixes do not appear in the suffix list are not touched. The suffix list is dot separated, and "*" wildcards work. Thus the suffix list:

".c.y.pay*.h"

allows *error* to touch files ending with ".c", ".y", ".pay*" and ".y".

−s Print out *statistics* regarding the error categorization (not too useful).

FILES

~/.errorrc	function names to ignore for *lint* error messages
/dev/tty	user terminal

RESTRICTIONS

Error opens the terminal directly to do user querying.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor format of error messages may cause *error* to not understand the error message.

Because it is purely mechanical, *error* does not filter out numerous subsequent errors caused by by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected (error puts them before). The alignment of the " | " marking the point of error is not correct.

Error is designed for work on terminal screens at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

SUPPORT STATUS

Supported.

NAME

ex − text editor

SYNOPSIS

ex [−] [−v] [−t *tag*] [−r] [−R] [+*command*] [−l] [−x] name ...

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*. *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi*. Generally, the editor uses far more of the capabilities of terminals than *ed* does, and uses the terminal capability data base *terminfo*(4) and the type of the terminal you are using from the variable TERM in the environment 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**) 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. Pressing **^D** causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just pressing return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

Ex gives you more 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 with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the phone, you can use the editor **recover** command to retrieve your work. This gets 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, filenames in the editor may be formed with full shell metasyntax. The metacharacter **%** is also available in forming filenames and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

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 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 which allows the editor to automatically supply leading white space to align text. You can then use the ^D key as a backtab and space and tab forward to align new code easily.

Miscellaneous useful features include an intelligent **join** (j) command which 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*.

OPTIONS

The following invocation options are interpreted by *ex*:

− 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*
Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files is printed.

−R Set *readonly* mode preventing accidentally overwriting the file.

+*command*
Begin editing by executing the specified editor search or positioning *command*.

−l Set **LISP** mode; indenting appropriately for lisp code. The () {} [[and]] commands in *vi* are modified to have meaning for *lisp*.

−x Set encryption mode; a key is prompted for allowing creation or editing of an encrypted file (see *crypt*(1)). This option is dependent on the encryption module being installed and linked on the system. Without the encryption module, text is unchanged.

The *name* argument indicates files to be edited.

EX STATES

Command
: Normal and initial state. Input prompted for by :. Your kill character cancels partial command.

Insert
: Entered by **a**, **i** and **c**. Arbitrary text may be entered. Insert is normally terminated by line having only . on it, or abnormally with an interrupt.

Visual
: Entered by **vi**, terminates with **Q** or **^**.

EX COMMAND NAMES AND ABBREVIATIONS

abbrev	**ab**	next	**n**	unabbrev	**una**
append	**a**	number	**nu**	undo	**u**
args	**ar**			unmap	**unm**
change	**c**	preserve	**pre**	version	**ve**
copy	**co**	print	**p**	visual	**vi**
delete	**d**	put	**pu**	write	**w**
edit	**e**	quit	**q**	xit	**x**
file	**f**	read	**re**	yank	**ya**
global	**g**	recover	**rec**	window	**z**
insert	**i**	rewind	**rew**	escape	**!**
join	**j**	set	**se**	lshift	**<**
list	**l**	shell	**sh**	print next	CR
map		source	**so**	resubst	**&**
mark	**ma**	stop	**st**	rshift	**>**
move	**m**	substitute	**s**	scroll	**^D**

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 var.
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option
set no*x*	disable option
set *x*=*val*	give value *val*
set	show changed options
set all	show all options
set *x*?	show value of option *x*

MOST USEFUL OPTIONS

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
lisp		( ) { } are s-exp's
list		print ^I for tab, $ at end
magic		. [* special in patterns

number	nu	number lines
paragraphs	para	macro names which start ...
redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
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
window		visual mode lines
wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

SCANNING PATTERN FORMATION

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

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/usr/lib/*/*	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	preservation directory

SEE ALSO

awk(1), ed(1), edit(1), grep(1), sed(1), vi(1), curses(3X), term(4), terminfo(4), crypt(1).
Text Editor (ex) in the Editing Guide.

WARNINGS AND RESTRICTIONS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

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

File input/output errors do not print a name if the command line '−' option is used.

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

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

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

SUPPORT STATUS

Supported. The *crypt* feature is available only in the United States.

NAME

expr − evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by a backslash. 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*
: adds or subtracts integer-valued arguments.

expr { *, /, % } *expr*
: multiplies, divides, or remainders 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 RESTRICTIONS below).

3. # A better representation of example 2.

expr //$a : '.*/\(.*\)'

The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. expr $VAR : '.*'

returns the number of characters in $VAR.

SEE ALSO

ed(1), sh(1).

EXIT CODE

As a side effect of expression evaluation, *expr* returns the following exit values:

0	if the expression is neither null nor 0
1	if the expression *is* null or 0
2	for invalid expressions.

DIAGNOSTICS

syntax error

for operator/operand errors

non-numeric argument

if arithmetic is attempted on such a string

RESTRICTIONS

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 are all taken as the = operator). The following works:

```
expr X$a = X=
```

SUPPORT STATUS

Supported.

NAME

f77 − Fortran 77 compiler

SYNOPSIS

f77 [options] files

DESCRIPTION

F77 is the UNIX Fortran 77 compiler; it accepts several types of *file* arguments:

Arguments whose names end with .f are taken to be Fortran 77 source programs; they are compiled and each object program is left in the current directory in a file with the same name as the source, with .o substituted for .f.

Arguments whose names end with .r or .e are taken to be RATFOR or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by *f77*, producing .o files.

In the same way, arguments whose names end with .c or .s are taken to be C or assembly source programs and are compiled or assembled, producing .o files.

OPTIONS

The following *options* have the same meaning as in *cc*(1) (see *ld*(1) for link editor options):

−c	Suppress link editing and produce .o files for each source file.
−p	Prepare object files for profiling (see *prof*(1)).
−O	Invoke an object-code optimizer.
−S	Compile the named programs and leave the assembler-language output in corresponding files whose names are suffixed with .s. (No .o files are created.)
−o*output*	Name the final output file *output*, instead of **a.out**.
−f	In systems without floating-point hardware, use a version of *f77* that handles floating-point constants and links the object program with the floating-point interpreter.
−g	Generate additional information needed for the use of *sdb*(1).

The following *options* are peculiar to *f77*:

−onetrip	Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)
−1	Same as **−onetrip**.
−66	Suppress extensions which enhance Fortran 66 compatibility.
−C	Generate code for run-time subscript range-checking.
−U	Treat upper and lower cases as separate. *F77* is normally a no-case language (i.e. a is equal to A).
−u	Make the default type of a variable *undefined*, rather than using the default Fortran rules.

−v Provide diagnostics for each process during compilations.

−w Suppress all warning messages.

−F Apply EFL and RATFOR preprocessor to relevant files, put the result in files with suffix **.of**. (No **.o** files are created.)

−m Apply the M4 preprocessor to each EFL or RATFOR source file before transforming with the *ratfor*(1) or *efl*(1) processors.

−E Use the remaining characters in the argument as an EFL flag argument whenever processing a .e file.

−R Use the remaining characters in the argument as a RATFOR flag argument whenever processing a .r file.

Other arguments are taken to be either link-editor option arguments or *f77*-compilable object programs (typically produced by an earlier run), or libraries of *f77*-compilable routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the default name **a.out**.

FILES

file.[fresc]	input file
file.o	object file
a.out	linked output
./fort[*pid*].?	temporary
/usr/lib/f77pass1	compiler
/usr/lib/f77pass2	pass2
/lib/c2	optional optimizer
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/lib/libc.a	C library

SEE ALSO

FORTRAN in the Programming Guide.
asa(1), cc(1), fsplit(1), ld(1), m4(1), prof(1), ratfor(1).

DIAGNOSTICS

The diagnostics produced by *f77* are self-explanatory. Occasional messages may be produced by the link editor *ld*(1).

SUPPORT STATUS

Not supported.

NAME

factor − factor a number

SYNOPSIS

factor [number]

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you enter a positive number less than 2^{56} (about 7.2×10^{16}) it factors the number and prints its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

The maximum time to factor is proportional to the square root of the number and occurs when the number is prime or the square of a prime.

DIAGNOSTICS

Ouch

Input out of range or invalid input.

SUPPORT STATUS

Supported.

NAME

file − determine file type

SYNOPSIS

file [−**c**] [−**f** ffile] [−**m** mfile] *name*

DESCRIPTION

File performs a series of tests on each *name* in an attempt to classify the type of the file. If *name* appears to be an ASCII file, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable **a.out** file, *file* prints the version stamp, provided it is greater than 0 (see *ld*(1)).

File uses the file **/etc/magic** to identify any file that contains a numeric or string constant that indicates its type. Commentary at the beginning of **/etc/magic** explains its format.

OPTIONS

−**f***ffile*
: The *ffile* file contains the names of the files to be examined.

−**m***mfile*
: The *mfile* file is an alternate *magic* file.

−**c** Check the *magic* file for format errors and suppress file typing.

FILES

/etc/magic

SUPPORT STATUS

Supported.

NAME

find − find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) seeking files that match a boolean *expression* written in the primaries given below.

In the descriptions, the argument *n* is used as a decimal integer where +*n* means more than *n*, −*n* means less than *n*, and *n* means exactly *n*.

In the descriptions for *atime*, *mtime* and *ctime*, the current date is 0.

−**name** *file*
: True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for [, ?, and *).

−**perm** *onum*
: True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared.

−**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 (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 entering **y**.

−print
Always true; prints the current path name.

−cpio *device*
True when the device specified by *device* can be opened and written to. This option writes the current file on *device* in *cpio* (4) format (5120 byte records). The current file is written out in *cpio* copy out mode (i.e. **cpio −o** with no options).

−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.

(*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):

- The negation of a primary (**!** is the unary *not* operator).
- Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
- Alternation of primaries (**−o** is the *or* operator).

EXIT CODE

The exit status of *find* indicates normal or abnormal termination on the command, *not* whether or not anything was found which satisfies *expression*.

EXAMPLES

To remove all files in . (dot) named ***.o** that have not been accessed for a week:

```
find . −name '*.o' −atime +7 −exec rm {} \;
```

To list all plain files that have not been modified for 31 days in the *dir* directory:

```
find dir −type f −mtime +31 −exec ls −ld "{}" \;
```

To find all files in the current directory which contain *word*:

```
find . −type f −print|xargs grep −n 'word'
```

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1), cpio(1), sh(1), test(1), stat(2), cpio(4), fs(4).

SUPPORT STATUS
Supported.

NAME

fsplit − split f77 or ratfor files

SYNOPSIS

fsplit options files

DESCRIPTION

Fsplit splits the named *file(s)* into separate files, with one procedure per file. A procedure includes *blockdata*, *function*, *main*, *program*, and *subroutine* program segments. Procedure *X* is put in file *X*.**f**, *X*.**r**, or *X*.**e** depending on the language option chosen, with the following exceptions: *main* is put in the file *MAIN*.[**efr**] and unnamed *blockdata* segments in the files *blockdataN*.[**efr**] where *N* is a unique integer value for each file.

OPTIONS

If no options are chosen, the language is assumed to be f77.

−f Denotes that input files are *f77*.

−r Denotes that input files are *ratfor*.

−e Denotes that input files are *Efl*.

−s Strips *f77* input lines to 72 or fewer characters with trailing blanks removed.

SEE ALSO

csplit(1), f77(1), ratfor(1), split(1).

SUPPORT STATUS

Not supported.

NAME

hpd, erase, hardcopy, tekset, td − graphical device routines and filters

SYNOPSIS

hpd [−options] [GPS file ...]
erase
hardcopy
tekset
td [−eurn] [GPS file ...]

DESCRIPTION

All of the commands described below reside in **/usr/bin/graf** (see *graphics*(1G)).

hpd Hpd translates a GPS (see *gps*(4)), to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in *file* unless the **−u** or **−r** *option* is provided. If no *file* is given, the standard input is assumed. *Options* are:

- **c***n* Select character set *n*, *n* between 0 and 5 (see the *HP7221A Plotter Operating and Programming Manual, Appendix A*).
- **p***n* Select pen numbered *n*, *n* between 1 and 4 inclusive.
- **r***n* Window on GPS region *n*, *n* between 1 and 25 inclusive.
- **s***n* Slant characters *n* degrees clockwise from the vertical.
- **u** Window on the entire GPS universe.
- **xd***n* Set x displacement of the lower left corner of the viewport to *n* inches.
- **xv***n* Set width of viewport to *n* inches.
- **yd***n* Set y displacement of the lower left corner of the viewport to *n* inches.
- **yv***n* Set height of viewport to *n* inches.

erase *Erase* sends characters to a TEKTRONIX 4010 series storage terminal to erase the screen.

hardcopy When issued at a TEKTRONIX display terminal with a hard copy unit, *hardcopy* generates a screen copy on the unit.

tekset *Tekset* sends characters to a TEKTRONIX terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td *Td* translates a GPS to scope code for a TEKTRONIX 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in *file* unless the **−u** or **−r** *option* is provided. If no *file* is

given, the standard input is assumed. Options are:

e Do not erase screen before initiating display.

r*n* Display GPS region *n*, *n* between 1 and 25 inclusive.

u Display the entire GPS universe.

SEE ALSO

ged(1G), graphics(1G), gps(4).

SUPPORT STATUS

Not supported.

NAME

ged − graphical editor

SYNOPSIS

ged [−**euRrn**] [GPS file ...]

DESCRIPTION

Ged is an interactive graphical editor used to display, construct, and edit GPS files on TEKTRONIX 4010 series display terminals. If a GPS file(s) is specified, *ged* reads it into an internal display buffer and displays the buffer. The GPS in the buffer can then be edited. If − is given as a file name, *ged* reads a GPS from the standard input.

A GPS file is composed of instances of three graphical objects: *lines*, *arc*, and *text*. *Arc* and *lines* objects have a start point, or *object-handle*, followed by zero or more points, or *point-handles*. *Text* has only an object-handle. The objects are positioned within a Cartesian plane, or *universe*, having 64K (−32K to +32K) points, or *universe-units*, on each axis. The universe is divided into 25 equal sized areas called *regions*. Regions are arranged in five rows of five squares each, numbered 1 to 25 from the lower left of the universe to the upper right.

Ged maps rectangular areas, called *windows*, from the universe onto the display screen. Windows allow the user to view pictures from different locations and at different magnifications. The *universe-window* is the window with minimum magnification, i.e. the window that views the entire universe. The *home-window* is the window that completely displays the contents of the display buffer.

OPTIONS

Ged accepts the following command line options:

- **e** Do not erase the screen before the initial display.
- **r***n* Display region number *n*.
- **u** Display the entire GPS *universe*.
- **R** Restricted shell invoked on use of !.

COMMANDS

Ged commands are entered in *stages*. Typically each stage ends with a <cr> (return). Prior to the final <cr> the command may be aborted by pressing **rubout**. The input of a stage may be edited during the stage using the erase and kill characters of the calling shell. The prompt * indicates that *ged* is waiting at stage 1.

Each command consists of a subset of the following stages:

1. *Command line*
   A command *name* followed by *argument*(s) followed by a <cr>. A command *name* is a single character. Command *arguments* are either *option*(s) or a *filename*. *Options* are indicated by a leading −.
2. *Text* A sequence of characters terminated by an unescaped <cr>. (120 lines of text maximum.)

3. *Points* A sequence of one or more screen locations (maximum of 30) indicated either by the terminal crosshairs or by name. The prompt for entering *points* is the appearance of the crosshairs. When the crosshairs are visible, entering:

   sp (space) enters the current location as a *point*. The *point* is identified with a number.

   enters the previous *point* numbered *n*.

   >*x* labels the last *point* entered with the upper case letter *x*.

   enters the *point* labeled *x*.

   . establishes the previous *points* as the current *points*. At the start of a command the previous *points* are those locations given with the previous command.

   = echoes the current *points*.

   n enters the *point* numbered *n* from the previous *points*.

   # erases the last *point* entered.

   @ erases all of the *points* entered.

4. *Pivot* A single location, entered by pressing <cr> or by using the operator, and indicated with a *.

5. *Destination*
   A single location entered by pressing <cr> or by using

COMMAND SUMMARY

In the summary, characters entered by the user are printed in **bold**. Command stages are printed in *italics*. Arguments surrounded by brackets ([]) are optional. Parentheses surrounding arguments separated by **or** means that exactly one of the arguments must be given.

Construct commands:

Arc	[—echo,style,weight] *points*
Box	[—echo,style,weight] *points*
Circle	[—echo,style,weight] *points*
Hardware	[—echo] *text points*
Lines	[—echo,style,weight] *points*
Text	[—angle,echo,height,mid-point,right-point,text,weight] *text points*

Edit commands:

Delete	(− (universe or view) or *points*)
Edit	[−angle,echo,height,style,weight] (− (universe or view) or *points*)
Kopy	[−echo,points,x] *points pivot destination*
Move	[−echo,points,x] *points pivot destination*
Rotate	[−angle,echo,kopy,x] *points pivot destination*
Scale	[−echo,factor,kopy,x] *points pivot destination*

View commands:

coordinates	*points*
erase	
new-display	
object-handles	(− (universe or view) or *points*)
point-handles	(− (labeled-points or universe or view) or *points*)
view	(− (home or universe or region) or [−x] *pivot destination*)
x	[−view] *points*
zoom	[−out] *points*

Other commands:

quit or Quit	
read	[−angle,echo,height,mid-point,right-point,text,weight] *file-name* [*destination*]
set	[−angle,echo,factor,height,kopy,mid-point,points, right-point,style,text,weight,x]
write	*file-name*
!*command*	
?	

Command options:

Options specify parameters used to construct, edit, and view graphical objects. If a parameter used by a command is not specifed as an *option*, the default value for the parameter is used (see set below). The format of command *options* is

−*option*[,*option*]

where *option* is *keyletter*[*value*]. Flags take on the *values* of true or false indicated by + and − respectively. If no *value* is given with a flag, true is assumed.

Object options:

angle*n*	Angle of *n* degrees.
echo	When true, echo additions to the display buffer.
factor*n*	Set scale factor to *n* percent.
height*n*	Set height of *text* to *n* universe-units ($0 \leq n < 1280$).
kopy	When true, copy rather than move.
mid-point	When true, use mid-point to locate text string.
points	When true, operate on points; otherwise operate on objects.
right-point	When true, use right-point to locate *text* string.
style*type*	Set line style to one of following *types*: **so** solid **da** dashed **dd** dot-dashed **do** dotted **ld** long-dashed
text	When false, outline *text* strings rather than draw them.
weight*type*	Set line weight to one of following *types*: **n** narrow **m** medium **b** bold

Area options:

home	Reference the home-window.
out	Reduce magnification.
region*n*	Reference region *n*.
universe	Reference the universe-window.
view	Reference those objects currently in view.
x	Indicate the center of the referenced area.

COMMAND DESCRIPTIONS

Construct commands:

Arc and Lines

behave similarly. Each consists of a *command line* followed by *points*. The first *point* entered is the object-handle. Successive *points* are point-handles. Lines connect the handles in numerical order. Arc fits a curve to the handles. Currently, a maximum of 3 points fit with a circular arc.

Box and Circle

are special cases of Lines and Arc, respectively. Box generates a rectangle with sides parallel to the universe axes. A diagonal of the rectangle connects the first *point* entered with the last *point*. The first *point* is the object-handle. Point-handles are created at each of the vertices. Circle generates a circular arc centered about the *point* numbered zero and

passing through the last *point*. The circle's object-handle coincides with the last *point*. A point-handle is generated 180 degrees around the circle from the object-handle.

Text and Hardware
generate *text* objects. Each consists of a *command line*, *text* and *points*. *Text* is a sequence of characters delimited by <cr>. Multiple lines of text may be entered by preceding a cr with a backslash. The Text command creates software generated characters. Each line of software text is treated as a separate *text* object. The first *point* entered is the object-handle for the first line of text. The Hardware command sends the characters in *text* uninterpreted to the terminal.

Edit commands:

Edit commands operate on portions of the display buffer called *defined areas*. A defined area is referenced either with an area *option* or interactively. If an area *option* is not given, the perimeter of the defined area is indicated by *points*. If no *point* is entered, a small defined area is built around the location of the <cr>. This is useful to reference a single *point*. If only one *point* is entered, the location of the <cr> is taken in conjunction with the *point* to indicate a diagonal of a rectangle. A defined area referenced by *points* is outlined with dotted lines.

Delete
removes all objects whose object-handle lies within a defined area. The universe option removes all objects and erases the screen.

Edit
modifies the parameters of the objects within a defined area. Parameters that can be edited are:

angle	angle of *text*
height	height of *text*
style	style of *lines* and *arc*
weight	weight of *lines*, *arc*, and *text*.

Kopy (or Move)
copies (or moves) object- and/or point-handles within a defined area by the displacement from the *pivot* to the *destination*.

Rotate
rotates objects within a defined area around the *pivot*. If the kopy flag is true then the objects are copied rather than moved.

Scale
scales point displacements from the pivot by factor percent for objects whose object-handles are within a defined area. If the kopy flag is true then the objects are copied rather than moved.

View commands:

coordinates
prints the location of *point*(s) in universe- and screen-units.

erase
clears the screen (but not the display buffer).

new-display
erases the screen then displays the display buffer.

object-handles (or point-handles)
labels object-handles (and/or point-handles) that lie within the defined area with **O** (or **P**). point-handles identifies labeled points when the labeled-points flag is true.

view
moves the window so that the universe point corresponding to the *pivot* coincides with the screen point corresponding to the *destination*. Options for home, universe, and region display particular windows in the universe.

x
indicates the center of a defined area. Option view indicates the center of the screen.

zoom
decreases or increases the magnification of the viewing window based on the defined area. For increased magnification, the window is set to circumscribe the defined area. For a decrease in magnification the current window is inscribed within the defined area.

Other commands:

quit or Quit
exits from *ged*. Quit responds with ? if the display buffer has not been written since the last modification.

read
inputs the contents of a file. If the file contains a GPS it is read directly. If the file contains text it is converted into *text* object(s). The first line of a text file begins at *destination*.

set
when given *option*(s) resets default parameters, otherwise it prints current default values.

write
outputs the contents of the display buffer to a file.

! escapes *ged* to execute a UNIX system command.

? lists *ged* commands.

SEE ALSO

gdev(1G), graphics(1G), sh(1), gps(4).
Graphics Editor in the Programming Guide.

SUPPORT STATUS

Not supported.

NAME

get − get a version of an SCCS file

SYNOPSIS

get [−**r**SID] [−**c**cutoff] [−**i**list] [−**x**list] [−**w**string] [−aseq-no.] [−**k**] [−**e**] [−**l**[**p**]] [−**p**] [−**m**] [−**n**] [−**s**] [−**b**] [−**g**] [−**t**] file ...

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its optional arguments, which begin with −. The options may be specified in any order, but all options apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the standard input is assumed to be the name of an SCCS file to be processed.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading s.; (see also *FILES*, below).

OPTIONS

Each of the options is explained below as though only one SCCS file is to be processed, but the effects of any option apply independently to each named file.

−r*SID* Indicates the *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the −e option is also used), as a function of the SID specified.

−c*cutoff* Denotes the *cutoff* date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **−c7502** is equivalent to **−c750228235959**. Any number of non-numeric characters may separate the various two digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form:

−c77/2/2 9:22:25.

Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

~!get "−c%E% %U%" s.file

−e Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The −e option used in a *get* for a particular version (SID) of the SCCS file prevents further

gets for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of get **−e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an **−e** option is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the **−k** option in place of the **−e** option.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **−e** option is used.

−b Used with the **−e** option to indicate that the new delta should have an SID in a new branch as shown in Table 1. This option is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

−i*list* Includes a *list* of deltas in the creation of the generated file. The *list* has the following syntax:

<list> ::= <range> <list> , <range>
<range> ::= SID SID − SID

SID, the SCCS Identification of a delta, may be in any form shown in the *SID Specified* column of Table 1. Partial SIDs are interpreted as shown in the *SID Retrieved* column of Table 1.

−x*list* Excludes a *list* of deltas in the creation of the generated file. See the **−i** option above for the *list* format.

−k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **−k** option is implied by the **−e** option.

−l[p] Writes a delta summary into an *l-file*. If **−lp** is used then an *l-file* is not created; *get* writes the delta summary on the standard output instead. See *FILES* for the format of the *l-file*.

−p Writes the text retrieved from the SCCS file on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **−s** option is used, in which case it disappears.

−s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) are unaffected.

−m Precedes each text line retrieved from the SCCS file by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

−n Precedes each generated text line with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **−m** and **−n** options are used, the format is: %M% value, followed by a horizontal tab, followed by the **−m** option generated format.

−g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

−t Accesses the most recently created delta in a given release (e.g., **−r1**), or release and level (e.g., **−r1.2**).

−w*string* Substitute *string* for all occurances of %W% when *get*ing the file.

−a*seq-no.* Retrieves the delta sequence number of the SCCS file delta (version) (see *sccsfile*(5)). This option is used by the *comb*(1) command. If both the **−r** and **−a** options are specified, the **−a** option is used. Care should be taken when using the **−a** option in conjunction with the **−e** option, as the SID of the delta to be created may not be what one expects. The **−r** option can be used with the **−a** and **−e** options to control the naming of the SID of the delta to be created.

SCCS IDENTIFICATION STRING

For each file processed, *get* prints the SID being accessed and the number of lines retrieved from the SCCS file on the standard output.

If the **−e** option is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, *get* prints each file name (preceded by a new-line) before processing it. If the **−i** option is used, *get* lists included deltas following the notation *Included*; if the **−x** option is used, excluded deltas are listed following the notation *Excluded*.

TABLE 1. Determination of SCCS Identification String

SID[1] Specified	−b Keyletter Used[5]	Other Conditions	SID Retrieved	SID of Delta to be Created
none[6]	no	R defaults to mR	mR.mL	mR.(mL+1)
none[6]	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1[3]
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	—	R < mR and R does *not* exist	hR.mL[2]	hR.mL.(mB+1).1
R	—	Trunk succ.[4] in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	—	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	—	Branch succ.	R.L.B.S	R.L.(mB+1).1

1 **R**, **L**, **B**, and **S** are the *release* , *level*, *branch*, and *sequence* components of the SID, respectively; **m** means *maximum*. Thus, for example, **R.mL** means *the maximum level number within release* **R**; **R.L.(mB+1).1** means *the first sequence number on the new branch* (i.e., maximum branch number plus one) *of level* **L** *within release* **R**. Note that if the SID specified is of the form **R.L**, **R.L.B**, or **R.L.B.S**, each of the specified components *must* exist.

2 **hR** is the highest *existing* release that is lower than the specified, *nonexistent*, release **R**.

3 Forces creation of the *first* delta in a *new* release.

4 Successor.

5 The **−b** option is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of — means *irrelevant*.

6 Applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file (see *admin*(1)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see *admin*(1)).
%C%	Current line number. This option is intended for identifying messages output by the program. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string @(#) recognizable by *what*(1).
%W%	A shorthand notation for constructing *what*(1) strings for the UNIX system program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing *what*(1) strings for non-UNIX system program files. %A% = %Z%%Y% %M% %I%%Z%

FILES

Several auxiliary files may be created by *get*. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, for **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **−p** option is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **−k** option is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **−l** option is used; its mode is 444 and it is owned

by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
   * otherwise.
b. A blank character if the delta was applied or was not applied and ignored;
   * if the delta was not applied and was not ignored.
c. A code indicating a *special* reason why the delta was or was not applied:
   I: Included.
   X: Excluded.
   C: Cut off (by a −c option).
d. Blank.
e. SCCS identification (SID).
f. Tab character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. Login name of person who created *delta*.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an −e option along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an −e option for the same SID until *delta* is executed or the joint edit flag, j, (see *admin*(1)) is set in the SCCS file. *Get* creates the *p-file* in the directory containing the SCCS file; the effective user must have write permission in that directory. Its mode is 644 and the effective user owns it. The format of the *p-file* is: the retrieved SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the −i option argument if it was present, followed by a blank and the −x option argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1), sccsfile(4).
Source Code Control System User Guide in the Support Tools Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

RESTRICTIONS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the −e option is used.

SUPPORT STATUS

Supported.

NAME

getopt − parse command options

SYNOPSIS

set −− `getopt optstring $*`

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures and to check for valid 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 not used explicitly, *getopt* generates it; in either case, *getopt* places it at the end of the options. The positional parameters of the shell ($1 $2 ...) are reset so that each option is preceded by a − and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set −− `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        −a | −b)        FLAG=$i; shift;;
        −o)             OARG=$2; shift 2;;
        −−)             shift; break;;
        esac
done
```

This code accepts any of the following as equivalent:

```
cmd −aoarg file file
cmd −a −o arg file file
cmd −oarg −a file file
cmd −a −oarg −− file file
```

SEE ALSO

sh(1), getopt(3C).

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

SUPPORT STATUS

Supported.

NAME

graph − draw a graph

SYNOPSIS

graph [options]

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *tplot*(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with double-quotes, in which case they may be empty or contain blanks and numbers; labels never contain new-lines.

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.

OPTIONS

−a *sp st*

Supplies abscissas automatically (they are missing from the input). *Sp* is the spacing, and if not given is assumed to be 1. *St* is the starting point for automatic abscissas; *st* is assumed to be 0 if not specified, or the lower limit given by −x.

−b

Breaks (disconnects) the graph after each label in the input.

−c *string*

Denotes character string *string* as the label for each point.

−g *gstyle*

Denotes *gstyle* as the grid style: 0 for no grid, 1 for frame with ticks, and 2 for full grid. If this option is omitted, full grid is assumed.

−l *label*

Denotes *label* as the label for the graph.

−m *mode*

Denotes *mode* as the style of connecting lines: 0 for disconnected, and 1 for connected; connected is assumed. 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

Saves screen, does not erase before plotting.

−x [*l*] [[[*low*] *high*] *g*]

If *l* is present, x axis is logarithmic. *low* is the lower x limit; *high* is the upper x limit; *g* is the grid spacing on the x axis. Normally these quantities are determined automatically.

−y [*l*] [[[*low*] *high*] *g*]

Similar to −x, except for y axis.

−h *ht*
: Denotes *ht* as fraction of space for height.

−w *wd*
: Denotes *wd* as fraction of space for width.

−r *rt*
: Denotes *rt* as fraction of space to move right before plotting.

−u *up*
: Denotes *up* as fraction of space to move up before plotting.

−t
: Transposes horizontal and vertical axes. (Option −x now applies to the vertical axis.)

SEE ALSO

graphics(1G), spline(1G), tplot(1G).
Graphics Overview in the Programming Guide.

RESTRICTIONS

Graph stores all points internally and drops those for which there is no room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

SUPPORT STATUS

Not supported.

NAME

graphics − access graphical and numerical commands

SYNOPSIS

graphics [−r]

DESCRIPTION

Graphics prefixes the path name **/usr/bin/graf** to the current **$PATH** value, changes the primary shell prompt to **^**, and executes a new shell. The directory **/usr/bin/graf** contains all of the Graphics subsystem commands. If the **−r** option is given, access to the graphical commands is created in a restricted environment; that is, **$PATH** is set to

/:/usr/bin/graf:rbin:/usr/rbin

and the restricted shell, *rsh*, is invoked. To restore the environment that existed prior to issuing the *graphics* command, enter **EOT** (control-d on most terminals). To logoff from the graphics environment, enter **quit**.

The command line format for a command in *graphics* is *command name* followed by *argument*(s). An *argument* may be a *file name* or an *option string*. A *file name* is the name of any UNIX system file except those beginning with −. The *file name* − is the name for the standard input. An *option string* consists of − followed by one or more *option*(s). An *option* consists of a keyletter possibly followed by a value. *Options* may be separated by commas.

The graphical commands are partitioned into four groups.

Commands that manipulate and plot numerical data; see *stat*(1G).

Commands that generate tables of contents; see *toc*(1G).

Commands that interact with graphical devices; see *gdev*(1G) and *ged*(1G).

A collection of graphical utility commands; see *gutil*(1G).

A list of the *graphics* commands can be generated by entering **whatis** in the *graphics* environment.

SEE ALSO

gdev(1G), ged(1G), gutil(1G), stat(1G), toc(1G), gps(4).
Graphics Overview in the Programming Guide.

SUPPORT STATUS

Not supported.

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 (which is the *nroff*(1) default terminal) for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, *greek* attempts to use the environment variable **$TERM** (see *environ*(5)). 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), eqn(1), hp(1), mm(1), tplot(1G), nroff(1), environ(5), term(5).

SUPPORT STATUS

Not supported.

NAME

grep, egrep, fgrep − search a file for a pattern

SYNOPSIS

grep [options] expression [files]

egrep [options] [expression] [files]

fgrep [options] [strings] [files]

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. A *grep* pattern is a limited regular *expression* in the style of *ed*(1); *grep* uses a compact nondeterministic algorithm. An *egrep* pattern is a full regular *expression*; *egrep* uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; *fgrep* is fast and compact.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters $, *, [, ^, |, (,), and \ in *expression* because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes.

Fgrep searches for lines that contain one of the *strings* separated by newlines.

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

- A regular expression followed by + matches one or more occurrences of the regular expression.
- A regular expression followed by ? matches 0 or 1 occurrences of the regular expression.
- Two regular expressions separated by | or by a newline match strings that are matched by either.
- A regular expression may be enclosed in parentheses () for grouping.

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

OPTIONS

−v Print all lines but those matching.

−x Print only lines matched in their entirety (*fgrep* only).

−c Print only a count of matching lines.

−i Ignore upper/lower case distinctions during comparisons.

−l List only the names of files with matching lines (once), separated by newlines.

−n Precede each line with its relative line number in the file.

−b Precede each line with the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

−s Suppress the error messages produced for nonexistent or unreadable files (*grep* only).

−e *expression*
Same as a simple *expression* argument, but useful when the

expression begins with a − (does not work with *grep*).

−f *file*
Take the regular *expression* (*egrep*) or *strings* list (*fgrep*) from the *file*.

EXAMPLE

To find all lines in all files in the current directory which contain *word* enter

```
find . −type f −print|xargs grep −n 'word'
```

SEE ALSO

ed(1), sed(1), sh(1).

DIAGNOSTICS

The exit code is 0 if one or more matches are found, 1 if no matches are found, or 2 if there are syntax error(s) or inaccessible file(s) even if matches are found.

RESTRICTIONS

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.

Egrep does not recognize ranges, such as [a−z], in character classes.

If there is a line with embedded nulls, *grep* only matches up to the first null. If it matches, the entire line is printed.

SUPPORT STATUS

Supported.

NAME

gutil − graphical utilities

SYNOPSIS

command-name [options] [files]

DESCRIPTION

Below is a list of miscellaneous device independent utility commands found in **/usr/bin/graf**. If no *files* are given, input is from the standard input. All output is to the standard output. Graphical data is stored in GPS format; see *gps*(4).

bel

Bel sends the bell character to the terminal

cvrtopt [=s*string* f*string* i*string* t*string*] [*args*]

Cvrtopt reformats *args* to facilitate processing by shell procedures. The *args* are usually the command line arguments of a calling shell procedure, and are either file names or option strings. A file name is a string not beginning with a hyphen, or a hyphen by itself; an option string is a string of options beginning with a hyphen.

Cvrtopt breaks apart the option string(s) into separate arguments and places the file names behind the options. Output is of the form:

−option −option . . . file name(s)

Options that take values (e.g., −r1.1) or are two letters long must be described through options to *cvrtopt.*

Cvrtopt is usually used with *set* in the following manner as the first line of a shell procedure:

set − `cvrtopt =**[*options*] **$@`

In the options to *cyrtopt* below, *string* is a one or two letter option name.

s*string*
: *String* accepts string values.

f*string*
: *String* accepts floating point numbers as values.

i*string*
: *String* accepts integers as values.

t*string*
: *String* is a two letter option name that takes no value.

gd [GPS *files*]

Gd (GPS dump) prints a readable listing of GPS.

gtop [−r*n***u**] [GPS *files*]

Gtop (GPS to *plot*(4) filter) transforms a GPS into *plot*(4) commands displayable by *plot* filters. GPS objects are translated if they fall within the window that circumscribes the first *file* unless an *option* is given.

Options:

r*n* translate objects in GPS region *n*.

u translate all objects in the GPS universe.

pd [*plot files*]

Pd (*plot*(4) dump) prints a readable listing of *plot*(4) format graphical commands.

ptog [*plot files*]

Ptog (*plot*(4) to GPS filter) transforms *plot*(4) commands into a GPS.

quit

Terminates a session.

remcom [*files*]

Remcom (remove comments) copies its input to its output with comments removed. Comments are as defined in C (i.e., /* comment */).

whatis [**−o**] [*names*]

Whatis prints a brief description of each *name* given. If no *name* is given, then the current list of description *names* is printed. **whatis** * prints out every description.

Option:

o just print command options

yoo *file*

Yoo is a piping primitive that deposits the output of a pipeline into a *file* used in the pipeline. Note that, without *yoo*, this is not usually successful as it causes a read and write on the same file simultaneously.

SEE ALSO

graphics(1G), gps(4).

SUPPORT STATUS

Not supported.

NAME

head − give first few lines

SYNOPSIS

head [−count] [file ...]

DESCRIPTION

Head is a filter which gives the first *count* lines of each of the specified files, or of the standard input.

If *count* is omitted it defaults to 10.

SEE ALSO

tail(1)

SUPPORT STATUS

Supported.

NAME

help − ask for help

SYNOPSIS

help [args]

DESCRIPTION

Help prints information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* prompts for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names. The arguments are of one of the following types.

type 1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., ge6, for message 6 from the *get* command).

type 2 Does not contain numerics (as a command, such as get)

type 3 Is all numeric (e.g., 212)

Help prints explanatory information related to the argument, if there is any.

If the *help* is inadequate, type in the command **help stuck** for information on further sources of help.

FILES

/usr/lib/help
directory containing files of message text.

/usr/lib/help/helploc
file containing locations of help files not in **/usr/lib/help**.

DIAGNOSTICS

Use *help*(1) for explanations.

SUPPORT STATUS

Supported.

NAME

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

SYNOPSIS

hp [−e] [−m]

DESCRIPTION

Hp supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most *nroff* output. A typical use is:

nroff −h files ... | hp

Regardless of the hardware options on your terminal, *hp* tries to do sensible things with underlining and reverse line-feeds. If the terminal has the *display enhancements* feature, subscripts and superscripts can be indicated in distinct ways. If it has the *mathematical-symbol* feature, Greek and other special characters can be displayed.

Hp provides the same set of Greek and other special characters, as does *300*(1), except that the *not* symbol is approximated by a right arrow, and only the top half of the integral sign is shown. The display is adequate for examining output from *neqn*.

OPTIONS

−e Assumes that your terminal has the *display enhancements* feature, and so makes maximal use 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 option 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.

DIAGNOSTICS

line too long

if the representation of a line exceeds 1,024 characters.

EXIT CODES

0 Normal termination.

2 Error.

SEE ALSO

300(1), col(1), eqn(1), greek(1), nroff(1), tbl(1).

RESTRICTIONS

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 erase text; in particular, tables generated by *tbl*(1) that contain vertical lines are often 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.

SUPPORT STATUS

Not supported.

NAME

id − print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

Id writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

SEE ALSO

logname(1), getuid(2).

SUPPORT STATUS

Supported.

NAME

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

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

Ipcrm removes one or more specified message, 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, and, after the last detach, destroys the shared memory segment and data structure associated with it.

−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, and, after the last detach, destroys the shared memory segment and data structure associated with it.

−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).

RESTRICTIONS

Message queues, shared memories, and semaphores created with IPC-PRIVATE keys can be removed by identifier only (i.e. **−q** *msqid*); they cannot be removed by key.

SEE ALSO

ipcs(1), msgctl(2), msgget(2), msgsnd(2), msgrcv(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

SUPPORT STATUS

Supported.

NAME

ipcs − report inter-process communication facilities status

SYNOPSIS

ipcs [options]

DESCRIPTION

Ipcs prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the options below. If any of the options **−q**, **−m**, or **−s** are specified, *ipcs* prints only the indicated information. If none of these three are specified, *ipcs* prints information about all three.

OPTIONS

−q Print information about active message queues.

−m Print information about active shared memory segments.

−s Print information about active semaphores.

−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 COLUMNS.

−c Print login name and group name of creator. See COLUMNS.

−o Print information on outstanding usage: number of messages on queue and total number of bytes in messages on queue for message queues, and number of processes attached to shared memory segments.

−p Print process number information: process ID of last process to send a message and process ID of last process to receive a message on message queues, and process ID of creating process and process ID of last process to attach or detach on shared memory segments. See COLUMNS.

−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 COLUMNS.

−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 is taken as the name of an alternate *namelist* (**/unix** is the default).

COLUMNS

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 are 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 is removed 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 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

IPCS(1)

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 namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgrcv(2), msgsnd(2), semop(2), shmop(2).

RESTRICTIONS

System status can change while *ipcs* is running; the information it gives is only a close approximation to the current status.

SUPPORT STATUS

Supported.

NAME

join − relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is −, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab or new-line. In this case, multiple separators count as one, and leading separators are ignored. The default output field separator is a blank.

OPTIONS

Some of the options use the argument *n*. This argument is 1 or 2 referring to either *file1* or *file2* respectively.

−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*
Form each output line from the fields specified in *list*; each element of the list 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 joins the password file and the group file, matching on the numeric group ID. The output is the login name, the group name, and the login directory. It is assumed that the files are 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).

RESTRICTIONS

With default field separation, the collating sequence is that of **sort −b**; with **−t**, the sequence is that of a plain sort.

Filenames that are numeric may cause conflict when the −o option is used right before listing filenames.

SUPPORT STATUS

Supported.

NAME

kill − terminate a process

SYNOPSIS

kill [−signo] process-id ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified process. This normally kills a process that does not catch or ignore the signal. The killed process must belong to the current user unless the current user is the superuser.

The process identification 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 identification of the last process in the pipeline is reported). Process identifications can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process identification number 0 is specified, all processes in the process group are signaled.

If a signal number preceded by − is given as the first argument, that signal is sent instead of terminate (see *signal*(2)). In particular **kill** −9 ... is a sure kill because the −9 signal cannot be caught or ignored by a process.

EXAMPLE

To kill processes, determine the process identifications by using *ps*(1) and then enter (assume the process identifications are 2336 and 2760):

```
kill 2336 2760
```

SEE ALSO

ps(1), sh(1), kill(2), signal(2).

SUPPORT STATUS

Supported.

NAME

last − indicate last logins of users and teletypes

SYNOPSIS

last [−N] [name ...] [tty ...]

DESCRIPTION

Last prints the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, *last* so indicates.

Last prints information from the *wtmp* file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes.

If *last* is interrupted, it indicates how far the search has progressed in *wtmp*.

Control-d (EOF) signal does nothing. Therefore, exit gracefully from *last* with a break or shift/delete signal.

ARGUMENTS

Arguments specify names of users or teletypes of interest.

Names of teletypes may be given fully or abbreviated. For example **last 00** is the same as **last tty00**.

If multiple arguments are given, *last* prints the information which applies to any of the arguments. For example

last root ttyb

would list all of the sessions of **root** as well as all sessions on the terminal **ttyb**.

Last with no arguments prints a record of all logins and logouts, in reverse order. The −N option limits the report to N lines.

EXAMPLE

The pseudo-user **reboot** logs in at reboots of the system, thus

last reboot

gives an indication of mean time between reboots.

FILES

/etc/wtmp login data base

SEE ALSO

utmp(4)

SUPPORT STATUS

Supported.

NAME

ld − link editor for common object files

SYNOPSIS

ld [−**a**] [−**e** epsym] [−**f** fill] [−**l**x] [−**m**] [−**r**] [−**s**] [−**o** outfile] [−**u** symname] [−**L** dir] [−**N**] [−**n**] [−**R** relocation address] [−**X**] [−**V**] [−**VS** num] filenames

DESCRIPTION

The *ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and *ld* combines them, producing an object module that can either be executed or used as input for a subsequent *ld* run. The output of *ld* is left in *a.out*. This file is executable if no errors occur during the load. If any input file, *filename*, is not an object file, *ld* assumes it is either a text file containing link editor directives or an archive library.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The library (archive) symbol table (see *ar*(4)) is searched sequentially with as many passes as are necessary to resolve external references that can be satisfied by library members. Thus, the ordering of library members is unimportant.

If a relocation address is specified by using the *−R* option, memory is configured to start at this address instead of the default zero. This relocates all segments to the specified relocation address.

OPTIONS

The following options are recognized by *ld*.

−a Produce an absolute file; give warnings for undefined references. Relocation information is stripped from the output object file unless the **−r** option is given. The **−r** option is needed only when an absolute file should retain its relocation information (not the normal case). If neither **−a** nor **−r** is given, **−a** is assumed.

−e *epsym*
Set the entry point address for the output file to be that of the symbol *epsym*. Defaults for epsym are used if one is not specified. If the symbol **start%** is defined it is used, otherwise **main**. If neither of these are defined, the text segment start address is used.

−f *fill*
Set the default fill pattern for gaps within an output section as well as initialized **bss** sections. The argument *fill* is a two-byte constant.

−lx Search a library **lib**x**.a**, where *x* is up to nine characters. A library is searched when its name is encountered, so the placement of a **−l** is significant. By default, libraries are located in **/lib**.

−m Produce a map or listing of the input/output sections on the standard output. A map is also produced if no object or library modules are processed.

−o *outfile*
Produce an output object file by the name *outfile*. The name of the default object file is **a.out**.

−r Retain relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. Unless **−a** is also given, the link editor does not complain about unresolved references.

−s Strip line number entries and symbol table information from the output object file.

−u *symname*
Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library because initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

−L *dir*
Change the algorithm of searching for **lib***x***.a** to look in *dir* before looking in **/lib** and **/usr/lib**. This option is effective only if it precedes the **−l** option on the command line.

−N Put the data section immediately following the text in the output file and flag the output file as non-shared text object. This is the default mode.

−n Align the data section to an 8K boundary following the text in the output file and flag the output file as shared text object. This allows the operating system to share text sections in memory when a second instance of this object is concurrently loaded.

−R *<addr>*
Relocate the start address of memory to *<addr>* so that segment memory allocation starts at this address. This leaves an unallocatable hole in logical memory.

−X No functionality. *Ld* does not terminate on this option so that Release 2.x *ld* command lines are processed. The option has no functionality in this release.

−V Output a message giving information about the version of *ld* being used.

−VS *num*
Use *num* as a decimal version stamp identifying the *a.out* file that is produced. The version stamp is stored in the optional header.

ALIGNMENT

The following information about section alignment and memory management unit requirements should be considered at system installation.

The default section alignment action for *ld* on M68000 systems is to align the code (.text) and data (.data and .bss combined) separately to a boundary determined by the −n option. If −n is present on the command line, alignment is to an 8K boundary; otherwise it is to a long (4 byte) boundary. Because memory management unit requirements vary from system to system, this alignment is not always desirable. The version of *ld* for M68000 systems, therefore, provides a mechanism to allow the specification of different section alignments for each system.

When all input files are processed (and if no override is provided), *ld* searches the list of library directories (as with the −l option) for a file named **default.ld**. If this file is found, it is processed as an *ld* instruction file (or **ifile**). The **default.ld** file should specify the required alignment as outlined below. If it does not exist, the default alignment action is taken.

The **default.ld** file should appear as follows with *<alignment>* replaced by the alignment requirement in bytes:

```
SECTIONS {
        .text : {}
        GROUP ALIGN(<alignment>) : {
                        .data : {}
                        .bss  : {}
                        }
        }
```

For example, a **default.ld** file of the following form provides the same alignment as for shared text alignment (−n specified):

```
SECTIONS {
        .text : {}
        GROUP ALIGN(32768) : {
                        .data : {}
                        .bss  : {}
                        }
        }
```

To get alignment on 2K-byte boundaries, the following **default.ld** file is specified:

```
SECTIONS {
        .text : {}
        GROUP ALIGN(2048) : {
                        .data : {}
                        .bss : {}
                        }
        }
```

In the absence of a **default.ld** file, default memory allocation (allocation without the −R option) configures memory on a 16-bit system starting at address 0x8000 and extending to address 0x1000000 and on a 32-bit system starting at address 0 and extending to address 0xffffffff. The data and bss group is aligned to a 4 byte boundary. If the −n option is specfied, the data and bss group is aligned on a 32 K boundary.

For more information about the format of *ld* instruction files or the meaning of the commands, see *The Link Editor* in the Support Tools Guide.

Release 2.x objects can be linked with the loader transparently. Two transformations must be made by the loader to these objects to make them linkable with the libraries and objects in this release.

First, the symbol naming conventions for C symbols are changed requiring the loader to map Release 2.x symbol names during processing. This involves stripping the leading underscores and mapping symbols truncated to 7 characters in Release 2.x to a possibly longer symbol name. When linking Release 2.x objects, a warning message is issued to inform what symbol name a Release 2.x symbol name has been mapped to. If a Release 2.x symbol maps to more than one new symbol name the loader aborts with a fatal error.

Relocation entries for Release 2.x objects must also be mapped differenly.

Note that the linker supports both the Release 2.x archive format and the archive format for this release. This permits linking with Release 2.x libraries.

FILES

/lib
/usr/lib
a.out output file

SEE ALSO

as(1), cc(1), a.out(4), ar(4).
The Link Editor in the Support Tools Guide.

WARNINGS

Through its options and input directives, the link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives should insure the following properties for programs:

C defines a zero pointer as null. A pointer to which zero is assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the data space.

When the link editor is called through *cc*(1), a startup routine is linked with the user program. This routine calls exit () (see *exit*(2)) after execution of the main program. If the user calls the link editor directly, then the user must insure that the program always calls exit() rather than falling through the end of the entry routine.

When linking an application program, the text segment starts at address 0.

Link applications with the appropriate floating point C library. Use −lc for Release 2.x floating point support. Use −lcieee for this release floating point support. There are also corresponding math libraries for each floating point support mode. See *cc*(1).

SUPPORT STATUS
Supported.

NAME

lex − generate programs for simple lexical tasks

SYNOPSIS

lex [−rctvn] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text.

The source *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

The program *lex* generates is named *yylex()*, and the program *main()* contained in the library calls *yylex()*.

Lex generates a file *lex.yy.c* from the input files. *Lex.yy.c*, when loaded with the library, copies the input to the output, searching for a string specified in its rules. When a matching string is found, the corresponding program text in the rule is executed, and the actual matched string is placed in *yytext*, an external character array. The *lex.yy.c* program matches strings in the order in which the source files specified the pattern strings. (See **SPECIAL CHARACTERS** below for furthur information on pattern strings.)

Lex expects three subroutines to be defined as macros in the source files:

input() to read a character
unput(c) to replace a character read
output(c) to write a character

These subroutines are defined in terms of standard streams, but user-written subroutines in the source files will override the standard subroutines. *Input()* reads from *yyin* and *output(c)* writes to *yyout*, which, if not specified, default to *stdin* and *stdout*.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes %% it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a %%, as in *yacc*(1). Lines preceding %% which begin with a non-blank character define the string on the left to be a constant, equal to the the remainder of the line; this string constant can be used later by surrounding it with {}. Note that curly brackets do not imply parentheses; only string substitution is done. (see **EXAMPLE** below.)

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p *n*
number of positions is *n* (default 2500)

%n *n*
number of states is *n* (500)

%e *n*
number of parse tree nodes is *n* (1000)

%a *n*
number of transitions is *n* (2500)

The use of one or more of the above automatically implies the −v option, unless the −n option is used.

SPECIAL CHARACTERS

Any of the following special characters may be used as an unordinary symbol if it is within double quotes or preceded by a backslash.

[...]	Matches a character class; for example, [abx-z] matches a, b, x, y or z.
*	Matches any nonnegative number of the previous character or character class.
+	Matches any positive number of the previous character or character class. For example, [A-Za-z] matches a string of letters.
?	Matches zero or one occurence of the previous character or character class.
.	Matches all ASCII characters except new-line.
()	Denotes grouping.
|	Denotes alternation.
r{*d*,*e*}	Matches between *d* and *e* occurrences of a patterrn *r*. These brackets have higher precedence than |, but lower precedence than *, +, ?, and concatenation.
^...	When placed at the beginning of a pattern, only matches text immediately following a new-line.
...$	When placed at the end of a pattern, only matches text immediately preceding a new-line.
/	Matches the new-line character. Only the part of the matched text up to the new-line is placed in *yytext*, but any characters following the slash are part of the pattern which must be matched.

EXAMPLE

```
D       [0−9]
%%
if      printf("IF statement\n");
[a−z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
                        {
                        case '/': break;
                        case '*': unput('*');
```

```
                    default: go to loop;
                    }
            }
```

OPTIONS

The options must appear before the source files on the command line. Multiple files are treated as a single file. If no files are specified, standard input is used.

−r indicates RATFOR actions

−c indicates C actions (default)

−t writes the *lex.yy.c* program to standard output

−v provides a summary of the statistics of the machine generated

−n suppresses the **−v** summary (default)

SEE ALSO

yacc(1).
Lexical Analyzer Generator in the *Support Tools Guide*

RESTRICTIONS

The **−r** option is not fully operational.

SUPPORT STATUS

Supported.

NAME

line − read one line

SYNOPSIS

line

DESCRIPTION

Line copies one line (all characters 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 terminal of the user.

SEE ALSO

sh(1), read(2).

SUPPORT STATUS

Supported.

NAME

lint − a C program checker

SYNOPSIS

lint [option] ... file ...

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be erroneous, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things which are detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but not returned.

Arguments whose names end with .c are taken to be C source files. Arguments whose names end with **.ln** are taken to be the result of an earlier invocation of *lint* with either the **−c** or the **−o** option used. The **.ln** files are analogous to **.o** (object) files that are produced by the *cc*(1) command when given a .c file as input. Files with other suffixes are warned about and ignored.

Lint takes all the **.c**,**.ln**, and **llib-l***x***.ln** (specified by **−l***x*) files and processes them in their command line order. By default, *lint* appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **−p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **−c** option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the **−c** option is used, the **.ln** and the **llib-l***x***.ln** files are ignored.

The pre-processor symbol "lint" is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C source change the behavior of *lint*:

/*NOTREACHED*/
: at appropriate points stops comments about unreachable code. This comment is typically placed just after calls to functions like *exit*(2).

/*VARARGS*n*/
: suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/
: turns on the **−v** option for the next function.

/*LINTLIBRARY*/
: at the beginning of a file shuts off complaints about unused functions in this file. This is equivalent to using

the −v and −x options.

Lint produces its first output on a per source file basis. Complaints regarding included files are collected and printed after all source files are processed. If the −c option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name is printed followed by a question mark.

The behavior of the −c and the −o options allows for incremental use of *lint* on a set of C source files. Generally, one invokes *lint* once for each source file with the −c option. Each of these invocations produces a .ln file which corresponds to the .c file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the −c option), listing all the .ln files with the needed −l*x* options. This prints all the inter-file inconsistencies. This scheme works well with *make*(1); it allows *make* to be used to *lint* only the source files that have been modified since the last time the set of source files were *lint*ed.

OPTIONS

Any number of *lint* options may be used, in any order. The following options are used to suppress certain kinds of complaints:

−a Suppress complaints about assignments of long values to variables that are not long.

−b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* often result in a large number of such complaints.)

−h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

−u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is useful for running *lint* on a subset of files of a larger program.)

−v Suppress complaints about unused arguments in functions.

−x Do not report variables referred to by external declarations but never used.

The following options alter the behavior of *lint*:

−l*x* Include additional lint library **llib-l***x***.ln**. You can include a lint version of the math library **llib-lm.ln** by inserting **−lm** on the command line. This argument does not suppress the default use of **llib-lc.ln**. These lint libraries must be in the assumed directory. This option can be used to keep local lint libraries and is useful in the development of multi-file projects.

−n Do not check compatibility against either the standard or the portable lint library.

−p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

−c Cause *lint* to produce a **.ln** file for every .c file on the command line. These **.ln** files are the product of *lint*'s first pass only, and are not checked for inter-function compatibility.

−o lib

Cause *lint* to create a lint library with the name **llib-l***lib***.ln**. The **−c** option nullifies any use of the **−o** option. The lint library produced is the input that is given to *lint*'s second pass. The **−o** option simply causes this file to be saved in the named lint library. To produce a **llib-l***lib***.ln** without extraneous messages, use of the **−x** option is suggested. The **−v** option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint comments".

The **−D**, **−U**, and **−I** options of *cpp*(1) and the **−g** and **−O** options of *cc*(1) are also recognized as separate arguments. The **−g** and **−O** options are ignored, but, by recognizing these options, *lint*'s behavior is closer to that of the *cc*(1) command. Other options are warned about and ignored.

FILES

/usr/lib	the directory where the lint libraries specified by the **−l***x* option must exist
/usr/lib/lint[12]	first and second passes
/usr/lib/llib-lc.ln	declarations for C Library functions (binary format; source is in **/usr/lib/llib-lc**)
/usr/lib/llib-port.ln	declarations for portable functions (binary format; source is in **/usr/lib/llib-port**)
/usr/lib/llib-lm.ln	declarations for Math Library functions (binary format; source is in **/usr/lib/llib-lm**)
/usr/tmp/*lint*	temporaries

SEE ALSO

cc(1), cpp(1), make(1).

RESTRICTIONS

Exit(2), *longjmp*(3C), and other functions which do not return are not understood; this causes various inaccurate messages.

SUPPORT STATUS

Supported.

NAME

login − sign on

SYNOPSIS

login [name [env-var ...]]

DESCRIPTION

Login is used at the beginning of each terminal session and allows you to identify yourself to the system. *Login* may be invoked as a command, by the system when a connection is first established, or by the system when a previous user has logged off.

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 it is not supplied as an argument, and, if appropriate, your password. Echoing is turned off where possible during the typing of your password so it does not appear on the written record of the session.

At some installations, an option may be invoked that requires you to enter a second password. This occurs only for dial-up connections, and is prompted by the message **dialup password:**. Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login:

- accounting files are updated,
- the procedure **/etc/profile** is performed,
- the message-of-the-day, if any, is displayed,
- the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh*(1)) are initialized, and
- the file **.profile** in the working directory is executed if it exists. (The **.profile** file in your home directory may be edited to include setting environment variables and executing programs.)

These specifications are found in the **/etc/passwd** file entry for the user. If the name of the command interpreter is not in the password file, the default command interpreter, **/bin/sh** is used. If this field is *, a *chroot*(2) is done to the directory named in the directory field of the entry. At that point, *login* is executed again at the new level which must have its own root structure including **/bin/login** and **/etc/passwd**.

The basic *environment* (see *environ*(5)) is initialized to:

HOME=*your-login-directory*
PATH=:/usr/ucb:/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, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which 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*
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile
.profile	user's login profile

SEE ALSO

mail(1), newgrp(1), sh(1), su(1), passwd(4), profile(4), environ(5).

DIAGNOSTICS

Login incorrect
: The user name or the password cannot be matched.

No shell
: Consult your system administator.

Cannot open password file
: Consult your system administator.

No directory
: Consult your system administator.

No utmp entry
: You probably attempted to execute *login* as a command without using the shell internal *exec* command or you attempted to execute *login* from other than the initial shell. You must exec **login** from the lowest level shell.

SUPPORT STATUS

Supported.

NAME

logname − get login name

SYNOPSIS

logname

DESCRIPTION

Logname reports the login name of the user by returning the content 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).

SUPPORT STATUS

Supported.

NAME

look − find lines in a sorted list

SYNOPSIS

look [**−df**] string [file]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

The options **d** and **f** affect comparisons as in *sort*(1):

d *Dictionary* order: only letters, digits, tabs and blanks participate in comparisons.

f Fold. Upper case letters compare equal to lower case.

If no *file* is specified, */usr/dict/words* is assumed with collating sequence **−df**.

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)

RESTRICTIONS

Some special characters, such as parentheses, ampersands, and carats, are shell sensitive and therefore are not recognized as strings. These characters may be used, however, if preceded by a backslash.

SUPPORT STATUS

Supported.

NAME

lorder − find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive *files* (see *ar*(1)). The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second.

The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1).

Note that the link editor *ld*(1) is capable of multiple passes over an archive in the portable archive format (see *ar*(4)) and does not require that *lorder*(1) be used when building an archive. The usage of the *lorder*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

EXAMPLE

The following example builds a new library from existing .o files.

```
ar cr library `lorder *.o | tsort`
```

FILES

*symref, *symdef temporary files

SEE ALSO

ar(1), ld(1), tsort(1), ar(4).

RESTRICTIONS

Object files whose names do not end with .o, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

SUPPORT STATUS

Supported.

NAME

lp − send requests to an LP line printer

SYNOPSIS

lp [−**c**] [−**d**dest] [−**m**] [−**n**number] [−**o**option] [−**s**] [−**t**title] [−**w**] files

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name − stands for the standard input and may be supplied on the command line along with named *files*. The order in which *files* appear is the same order in which they are printed.

Lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see *cancel(1)*) or find the status (see *lpstat*(1)) of the request.

OPTIONS

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* are not copied, but are 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. Also, in the absence of the −**c** option, any changes made to the named *files* after the request is made but before it is printed are 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 is printed only on that specific printer. If *dest* is a class of printers, then the request is 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 are printed. No mail is sent upon normal completion of the print request unless -m is specified.

−**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 valid *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 terminal after the *files* are printed. If the user is not logged in, then mail is sent instead.

FILES

/usr/spool/lp/*

SEE ALSO

cancel(1), enable(1), lpstat(1), mail(1), accept(1M), lpadmin(1M), lpsched(1M).
LP Spooling System in the Superuser Guide.

SUPPORT STATUS

Not supported.

NAME

lpstat − print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

Lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *Lpstat* prints the status of such requests.

OPTIONS

Options may appear in any order and may be repeated and intermixed with other arguments. Some of the options below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

−u"user1, user2, user3"

The omission of a *list* following such options causes all information relevant to the option to be printed, for example:

lpstat −o

prints the status of all output requests.

−a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

−c[*list*] Print class names and their members. *List* is a list of class names.

−d Print the system default destination for *lp*.

−o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.

−p[*list*] Print the status of printers. *List* is a list of printer names.

−r Print the status of the LP request scheduler.

−s Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

−t Print all status information.

−u[*list*] Print status of output requests for users. *List* is a list of login names.

−v[*list*] Print the names of printers and the pathnames of the devices associated with them. *List* is a list of printer names.

FILES
/usr/spool/lp/*

SEE ALSO
enable(1), lp(1).

SUPPORT STATUS
Not supported.

NAME

ls − list contents of directory

SYNOPSIS

ls [**−abcCdfFgilmnopqrRstux**] [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 file or directory argument is given, the current directory is listed. When several file or directory arguments are given, the arguments are first sorted appropriately, but files 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 which files are listed across the page, separated by commas. 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* database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

OPTIONS

−a List all entries; usually entries whose names begin with a period (.) are not listed.

−b Force printing of non-graphic characters to be in the octal \ddd notation.

−c Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**−t**) or printing (**−l**).

−C Output in multi-column with entries sorted down the columns.

−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.

−f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l**, **−t**, **−s**, and **−r**, and turns on **−a**; the order is the order in which entries appear in the directory.

−F Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable.

−g The same as **−l**, except that the owner is not printed.

−i For each file, print the i-number in the first column of the report.

−l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file. If the file is a special file, the size field contains the major and minor device numbers rather than a size.

−m Output stream format.

−n The same as **−l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

−o The same as **−l**, except that the group is not printed.
−p Put a slash (/) after each filename if that file is a directory.
−q Force printing of non-graphic characters in file names as the character (?).
−r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
−R Recursively list subdirectories encountered.
−s Give size in blocks, including indirect blocks, for each entry.
−t Sort by time modified (latest first) instead of by name.
−u Use time of last access instead of last modification for sorting (with the **−t** option) or printing (with the **−l** option).
−x Output multi-column with entries sorted across rather than down the page.

OUTPUT FORMAT

The mode printed under the **−l** option consists of 10 characters that are interpreted as follows:

The first character is:

d if the entry is a directory;
b if the entry is a block special file;
c if the entry is a character special file;
p if the entry is a fifo (i.e., named pipe) special file;
− if the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, execute permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
− if the indicated permission is *not* granted.

The group execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user execute permission character is given as **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or **−**) is **t** if the 1000 (octal) bit of the mode is on; this is the save text or sticky bit. The indications of set-ID and 1000 bits of the mode are capitalized (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	to get user IDs for **ls −l** and **ls −o**.
/etc/group	to get group IDs for **ls −l** and **ls −g**.

LS(1)

/usr/lib/terminfo/* to get terminal information.

SEE ALSO

chmod(1), find(1).

RESTRICTIONS

Unprintable characters in file names may confuse the columnar output options.

SUPPORT STATUS

Supported.

NAME

ls − list contents of directory

SYNOPSIS

/usr/ucb/ls [−acCdfFgilLqrRstu1] name ...

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 normally sorted alphabetically. When no file or directory argument is given, *ls* lists the current directory. When several file or directory arguments are given, the arguments are first sorted appropriately, but *ls* processes files before directories and their contents. Files whose names begin with a period are not listed (see **−a** option).

OPTIONS

−a List all entries, including those entries whose names begin with a period (.) (normally not listed except for the superuser).

−c Use time of file creation for sorting or printing.

−C Force multi-column output; this is the default when output is to a terminal.

−d If argument is a directory, list only its name; often used with **−l** to get the status of a directory.

−f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l**, **−t**, **−s**, and **−r**, and turns on **−a**; the order is the order in which entries appear in the directory.

−F Cause directories to be marked with a trailing /, sockets with a trailing =, symbolic links with a trailing @, and executable files with a trailing *.

−g Include the group ownership of the file in a long output.

−i For each file, print the i-number (inode number) in the first column of the report.

−l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. If the file is a special file, the size field contains the major and minor device numbers. If the file is a symbolic link, *ls* prints the pathname of the linked-to file preceded by −>.

−L If argument is a symbolic link, list the file or directory which the link references, rather than the link itself.

−q Force printing of non-graphic characters in file names as the character ?; this is the default when output is to a terminal.

−r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

−R Recursively list subdirectories encountered.

−s Give size in kilobytes of each file.

−t Sort by time modified (latest first) instead of by name.

−u Use time of last access instead of last modification for sorting (with the **−t** option) and/or printing (with the **−l** option).

−1 Force one entry per line output format; this is the default when output is not to a terminal.

OUTPUT FORMAT

The mode printed under the −l option consists of 10 characters that are interpreted as follows:

The first character is:

- **d** if the entry is a directory;
- **b** if the entry is a block special file;
- **c** if the entry is a character special file;
- **l** if the entry is a symbolic link;
- **s** if the entry is a socket; or,
- **−** if the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, execute permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

- **r** if the file is readable;
- **w** if the file is writable;
- **x** if the file is executable;
- **−** if the indicated permission is *not* granted.

The group execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user execute permission character is given as **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or **−**) is **t** if the 1000 (octal) bit of the mode is on; this is the save text or sticky bit. The indications of set-ID and 1000 bits of the mode are capitalized (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd to get *user ids* for **ls −l**.
/etc/group to get *group ids* for **ls −g**.

RESTRICTIONS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as **ls −s** is much different than **ls −s | lpr**. However, not using this setting almost certainly limits usage of old shell scripts which used *ls*.

SUPPORT STATUS

Supported.

NAME

m4 − macro processor

SYNOPSIS

m4 [options] [files]

DESCRIPTION

M4 is a macro processor intended to be a pre-processor for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is −, the standard input is read. The processed text is written on the standard output.

OPTIONS

The options and their effects are as follows:

−D*name*[=*val*]
Defines *name* to *val* or to null if *val* is not specified.

−U*name*
undefines *name*.

The following options must appear before the file names and before any −D or −U options.

−e Operate interactively. Interrupts are ignored and the output is unbuffered.

−s Enable line sync output for the C preprocessor (#line ...)

−B*int*
Change the size of the push-back and argument collection buffers from the default of 4,096.

−H*int*
Change the size of the symbol table hash array from the default of 199. The size should be prime.

−S*int*
Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

−T*int*
Change the size of the token buffer from the default of 512 bytes.

SYNTAX

Macro calls have the form:

name(arg1,arg2, ..., argn)

The (must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is assumed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore; the first character may not be a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are assumed to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call function as if the expanded macro had been placed into the text in the first place. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define (*mname*, *arg1*, *arg2*, ...)
: defines *arg1* as the value of the macro *mname*. When *mname* is subsequently used, *m4* replaces each occurrence of $*n* in the replacement text (where *n* is a digit) with the *n*-th argument. **$0** is the name of the macro; missing arguments are replaced by the null string; $# is the number of arguments; $* is a list of all the arguments separated by commas; $@ is like $*, but each argument is quoted with the current quotes (see *changequote*).

undefine (*mname*)
: removes the definition of the macro named in its argument.

defn (*m1*, *m2*, *m3*, ...)
: returns the quoted definition of its argument(s). *Defn* is useful for renaming macros, especially built-ins.

pushdef (*mname*, *arg1*, *arg2*, ...)
: like *define*, but saves any previous definition.

popdef (*m1*, *m2*, ...)
: removes current definition of its argument(s), exposing the previous one, if any.

ifdef (*mname*, *val1*, *val2*)
: if *mname* is defined, returns *val1*; otherwise returns *val2*. If *val2* is not specified, *ifdef* returns null. The word *unix* is predefined on the UNIX system versions of *m4*.

shift (*arg1*, *arg2*, ...)
: returns all but *arg1*. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that is subsequently performed.

changequote (*lq* ,*rq*)
: changes the quote symbols to *lq* and *rq*. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (left and right single quotes).

changecom (*lm* ,*rm*)
: change left and right comment markers from the default # and new-line. If *lm* and *rm* are not specified, the comment mechanism is effectively disabled. If only *lm* is specified, the left marker becomes *lm* and the right marker becomes new-

line. If both *lm* and *rm* are specified, both markers are changed. Comment markers may be up to five characters long.

divert (*digit-string*)
m4 maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to *digit-string* argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert (*dstring1, dstring2, ...*)
causes immediate output of text from the output streams named as arguments, or from all output streams if no arguments are specified. Text may be undiverted into another output stream. Undiverting discards the text in the output stream(s) specified by the arguments.

divnum
returns the value of the current output stream.

dnl reads and discards characters up to and including the next new-line.

ifelse (*arg1, arg2, arg3* [, *arg4 ...*])
returns *arg3* if *arg1* is the same string as *arg2*, *arg1*, *arg2*, *arg3* must be given. If *arg1* is not equal to *arg2*, and if five or more *arg*s are specified, *ifelse* repeats, using *arg*s 4, 5, 6, and 7; otherwise, *ifelse* returns *arg4*, or, if *arg4* is not present, null.

incr (*arg*)
returns the value of *arg* incremented by 1. The value of *arg* is calculated by interpreting an initial digit-string as a decimal number.

decr (*arg*)
returns the value of *arg* decremented by 1.

eval (*expr, radix, digits*)
evaluates *expr* as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. *Radix* specifies the radix for the result; the default is 10. *Digits* may be used to specify the minimum number of digits in the result.

len (*string*)
returns the number of characters in *string*.

index (*string, pattern*)
returns the position in *string* where *pattern* begins (zero origin), or −1 if *pattern* does not occur in *string*.

substr (*string, start, length*)
returns a substring of *string*. *Start* is a zero origin number selecting the first character; *length* indicates the length of the substring. If *length* is not specified, *length* is assumed to be

large enough to extend to the end of the first string.

translit (*chars*, *set1*, *set2*)
transliterates the characters in *chars* from the set *set1* to the set *set2*. No abbreviations are permitted.

include (*fname*)
returns the contents of the file named *fname*.

sinclude (*fname*)
returns the contents of the file named *fname*, but does not print a message, except that it says if the file is inaccessible.

syscmd (*unixcmd*)
executes the UNIX system command *unixcmd*. No value is returned.

sysval
is the return code from the last call to *syscmd*.

maketemp (*string*)
fills in a string of XXXXX in *string* with the current process ID.

m4exit (*xcode*)
exits immediately from *m4*. *xcode*, if given, is the exit code; if not given *xcode* is assumed to be 0.

m4wrap (*arg*)
arg is pushed back at final EOF; example: m4wrap(`cleanup()')

errprint (*arg*)
prints *arg* on the diagnostic output file.

dumpdef (*item1*, *item2*, ...)
prints current names and definitions, for the named items, or for all *items* if no arguments are given.

traceon (*m1*, *m2*, ...)
with no arguments, turns on tracing for the all macros (including built-ins). Otherwise, turns on tracing for named macros, *m1*, *m2*, ..., etc.

traceoff (*m1*, *m2*, ...)
turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

SEE ALSO

cc(1), cpp(1).
The M4 Macro Processor in the Support Tools Guide.

SUPPORT STATUS

Supported.

NAME

68000, pdp11, u3b, u3b5, vax − provide truth value about your processor type

SYNOPSIS

86000

pdp11

u3b

u3b5

vax

DESCRIPTION

The following commands return a true value (exit code of 0) if you are on a processor that the command name indicates.

68000	True if you are on a TOWER.
pdp11	True if you are on a PDP-11/45 or PDP-11/70.
u3b	True if you are on a 3B 20 computer.
u3b5	True if you are on a 3B 5 computer.
vax	True if you are on a VAX-11/750 or VAX-11/780.

The commands that do not apply return a false (non-zero) value. These commands are often used within *make*(1) makefiles and shell procedures to increase portability.

SEE ALSO

make(1), sh(1), test(1), true(1).

SUPPORT STATUS

Supported.

NAME

mail, rmail, smail − send mail to users or read mail

SYNOPSIS

mail [**−epqr**] [**−f** file]

smail [**−epqr**] [**−f** file]

mail [**−t**] persons

smail [**−t**] persons

rmail [**−t**] persons

DESCRIPTION

Mail, without arguments, prints mail for a user, message-by-message, in last-in, first-out order. For each message, the user is prompted with a ?, and a line is read from the standard input to determine the disposition of the message:

<newline>	Go on to next message.
+	Same as <newline>.
d	Delete message and go on to next message.
p	Print message again.
−	Go back to previous message.
s [*files*]	Save message in the named *files* (**mbox** is default).
w [*files*]	Save message, without its header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons* (yourself is default).
q	Put undeleted mail back in the *mailfile* and stop.
EOT (control-d)	Same as **q**.
x	Put all mail back in the *mailfile* unchanged and stop.
!*command*	Escape to the shell to do *command*. (Not valid for *smail*)
*	Print a command summary.

Smail is linked to *mail* and works like *mail* except that it does not allow the !*command*. *Smail* is primarily used as a security feature to prevent an unauthorized user access to UNIX utilities, but allows them to read their mail.

The options alter the printing of the mail:

−e Do not print mail. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

−p Print all mail without prompting for disposition.

−q Terminate *mail* after interrupts. Normally an interrupt only causes the termination of the message being printed.

−r Print messages in first-in, first-out order.

−f*file*
Use *file* (e.g., **mbox**) instead of the default *mailfile*.

−t Place the *names* of all persons to whom the mail was sent on the postmark of the mail for *each* person. This allows all who receive mail to know who else received that letter.

When *persons* are named, *mail* takes the standard input up to an end-of-file (typically control-d) or up to a line consisting of just a

period and adds it to the *mailfile* for each person. The message is preceded by the name of the sender and a postmark. Lines that look like postmarks in the message, (i.e., **From ...**) are preceded with a >. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file **dead.letter** is saved to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated each time it is needed erasing the previous contents of **dead.letter**.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)). Everything after the first exclamation mark in *persons* is interpreted by the remote system. In particular, if *persons* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying **a!b!cde** as the name of the recipient causes the message to be sent to user **b!cde** on system **a**. System **a** interprets that destination as a request to send the message to user **cde** on system **b**. This might be useful, for instance, if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**. *Mail* does not use *uucp* if the remote system is the local system name (i.e., localsystem!user).

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 is preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which causes all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of the mail for one person to one machine in a multiple machine environment. In order for forwarding to work properly, the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

Rmail permits only the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

FILES

/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), uucp(1C), write(1).

RESTRICTIONS

Race conditions sometimes result in a failure to remove a lock file. After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

SUPPORT STATUS

Supported.

NAME

mailx − interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The *mailx* command provides a comfortable, flexible environment for sending and receiving messages electronically. *Mailx* provides:

- Commands − When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages.
- Tilde Escapes − When sending mail, *mailx* provides tilde escape commands to allow editing, reviewing, and other modification of the message as it is entered.
- Environment Variables − *Mailx* provides environment variables which may be set and unset to control prompts, editing, formatting, and disposition of mail.

Incoming mail is stored in a standard file for each user called the system *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; this location is controlled by the **MBOX** environment variable. Messages remain in this file until forcibly removed.

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. When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, *mailx* reads the message and stores 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.

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 **set** and **unset** commands.

At startup time, *mailx* reads commands from the system file **/usr/lib/mailx/mailx.rc**, if present, to initialize certain parameters, then reads commands from a private startup file **$HOME/.mailrc** for personalized variables. Most regular commands are valid inside a startup file. The most common use of a startup file is to set up initial display options and alias lists. The following commands are not valid in the startup file: **!**, **Copy**, **edit**, **followup**, **Followup**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, and **visual**. Any errors in the startup file cause the remaining lines in the file to be ignored.

On the *mailx* command line, *options* start with a hyphen (−) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* attempts to read messages from the *mailbox*.

Recipients listed on the *mailx* 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 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 *lpr*(1) for recording outgoing mail on paper. Alias groups are set by the alias command and are lists of recipients of any type.

OPTIONS

Command line options are:

−d Turn on debugging output. Not recommended.

−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 overriding the **record** environment variable, if set.

−H Print header summary only.

−i Ignore interrupts. See also the **ignore** environment variable.

−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 escape commands are disabled.

−s *subject*
Use *subject* as the message Subject line.

−u *user*
Read *user*'s *mailbox*. This is only effective if *user*'s *mailbox* is not read protected.

−U Convert *uucp*(1) style addresses to internet standards overriding the **conv** environment variable.

COMMANDS

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, **print** is assumed.

Each message is assigned a sequential number, and there is at any time the notion of a *current* message marked by a > in the header summary. Many commands take an optional list of messages (*msglist*) to operate on which defaults to the current message. A *msglist* is a list of message specifications separated by spaces which may include:

n Message number **n**.
. Current message.
^ First undeleted message.
$ Last message.
***** All messages.

n—m Inclusive range of message numbers.
user All messages from **user**.
/string All messages with **string** in the Subject line; case is 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.

The following is a complete list of *mailx* commands:

!*shell-command*
Escape to the shell. Also see the **SHELL** environment variable.
 # comment
Null command (comment). This is useful in *.mailrc* files.
=
Print the current message number.
?
Print a summary of commands.
alias *alias name ...*
group *alias name ...*
Declare an alias for the given names. The names are substituted when *alias* is used as a recipient. This is useful in the *.mailrc* file.
alternates *name ...*
Declare a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **alternates** prints the current list of alternate names. Also see the **allnet** environment variable.
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. This is otherwise equivalent to the save command.
Copy [*msglist*]
Save the specified messages in a file whose name is derived from the author of the message to be saved; do not mark the messages as saved. This is otherwise equivalent to the Save

command.

delete [*msglist*]

Delete messages from the *mailbox*. If the **autoprint** environment variable is set, the next message after the last one deleted is printed.

discard [*header-field* ...]

ignore [*header-field* ...]

Suppress 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 Print and Type 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. This is almost equivalent to a delete command followed by a print 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** environment variable is used to get the name of the editor. The default editor is *ed*(1).

exit

xit

Exit from *mailx* without changing the *mailbox*. No messages are saved in the *mbox* (also see the **quit** command).

file [*filename*]

folder [*filename*]

Quit from the current file of messages and read in the specified file. The default file is the current *mailbox*. Several special characters are recognized when used as file names with the following substitutions:

%	Current *mailbox*.
%**user**	*Mailbox* for **user**.
#	Previous file.
&	Current *mbox*.

folders

Print the names of the files in the directory set by the **folder** environment variable.

followup [*message*]

Respond to a message recording the response in a file whose name is derived from the author of the message. This overrides the **record** environment variable, if set. Also see the Followup, Save, and Copy commands and the **outfolder** environment variable.

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. Also see the **followup**, Save, and Copy

commands and the **outfolder** environment variable.

from [*msglist*]
Print the header summary for the specified messages.

group *alias name ...*
alias *alias name ...*
Declare an alias for the given names. The names are substituted when *alias* is used as a recipient. This is useful in the *.mailrc* file.

headers [*message*]
Print the page of headers which includes the message specified. The **screen** environment variable sets the number of headers per page. Also see the **z** command.

help
Print a summary of commands.

hold [*msglist*]
preserve [*msglist*]
Hold the specified messages in the *mailbox*.

if *s* | *r*
mail-commands
else
mail-commands
endif
Conditional execution, where *s* executes the following *mail-commands* up to an **else** or **endif** if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. This is useful in the *.mailrc* file.

ignore *header-field ...*
discard *header-field ...*
Suppress 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 **Print** and **Type** commands override this command.

list
Print all commands available. No explanation is given.

mail *name ...*
Mail a message to the specified users.

mbox [*msglist*]
Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See the **MBOX** environment variable for a description of this file. Also see the **exit** and **quit** 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 because the name would be taken as a command in the absence of a real command. See the discussion of *msglists* 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** environment variable. If the **page** environment variable is set, a form feed character is inserted after each message.

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 field overriding suppression of fields by the **ignore** command.

print [*msglist*]
type [*msglist*]
Print the specified messages. If the **crt** environment variable is set, the messages longer than the number of lines specified by the **crt** environment variable are paged through the command specified by the **PAGER** environment variable. The default command is *pg*(1).

quit
Exit from *mailx* storing messages that were read in *mbox* and unread messages in the *mailbox*. Messages 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 the **record** environment variable is set to a filename, the response is saved at the end of that file.

reply [*message*]
respond [*message*]
Reply to the specified message including all other recipients of the message. If the **record** environment variable is set to a filename, the response is saved at the end of that file.

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. Also see the **Copy**, **followup**, and **Followup** commands and the **outfolder** environment variable.

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 the **eepsave** environment variable is set. Also see the **exit** and **quit** commands.

set
set *name*

set *name=string*
set *name=number*
Define an environment variable called *name*. The variable may be given a null, string, or numeric value. Set by itself prints all defined environment variables and their values.

shell
Invoke an interactive shell. Also see the **SHELL** environment variable.

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** environment variable is set, it is taken as the number of lines to print. The default is 5.

touch [*msglist*]
Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it is placed in the *mbox* upon normal termination. See **exit** and **quit**.

Type [*msglist*]
Print [*msglist*]
Print the specified messages on the screen including all header fields overriding suppression of fields by the **ignore** command.

type [*msglist*]
print [*msglist*]
Print the specified messages. If the **crt** environment variable is set, the messages longer than the number of lines specified by the **crt** environment variable are paged through the command specified by the **PAGER** environment variable. The default command is *pg*(1).

undelete [*msglist*]
Restore the specified deleted messages. This only restores messages deleted in the current mail session. If the **autoprint** environment variable is set, the last message of those restored is printed.

unset *name ...*
Erase the specified environment variables. If any variable was imported from the execution environment (i.e., a shell variable), it cannot be erased.

version
Print 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** environment variable is used to get the name of the editor. The default editor is *vi*(1).

write [*msglist*] *filename*
Write the given messages on the specified file minus the header and trailing blank line. This is otherwise equivalent

to the save command.

xit
exit

Exit from *mailx* without changing the *mailbox*. No messages are saved in the *mbox* (also see quit).

z[+|−]

Scroll the header display forward or backward one screen-full. The number of headers displayed is set by the **screen** environment variable.

TILDE ESCAPES

In *input mode*, commands are recognized by the tilde escape character (~). The tilde escape character is the default; this character may be changed by the **escape** environment variable. Lines not treated as commands are taken as input for the message. The following commands may be entered only from *input mode* by beginning a line with the escape character.

~! *shell-command*

Escape to the shell.

~.

Terminate message input by simulating the end of file.

~: *mail-command*
~_ *mail-command*

Perform the command. This is valid only when sending a message while reading mail.

~?

Print a summary of tilde escapes.

~A

Insert the autograph string *Sign* into the message. Also see the **Sign** environment variable.

~a

Insert the autograph string **sign** into the message. Also see the **sign** environment variable.

~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. Also see the **DEAD** environment variable.

~e

Invoke the editor specified by the **EDITOR** environment variable on the partial message. The default editor is *ed*(1).

~f [*msglist*]

Forward the specified messages. The messages are inserted into the current message without alteration.

~h

Prompt for the 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.'

~m [*msglist*]
Insert the specified messages into the letter shifting the new text to the right one tab stop. This is 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 the **DEAD** environment variable 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 a shell command which is executed and the standard output is 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 the screen editor specified by the **VISUAL** environment variable on the partial message. The default editor is *vi*(1).

~w *filename*
Write the partial message onto the given file without the header.

~x
Exit from the *input mode* as with ~q except do not save the message 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. These variables are not alterable within *mailx*.

HOME=*directory*
The user's home directory during execution of *mailx*.

MAILRC=*filename*
The name of the startup file executed during startup. The default is $HOME/.mailrc.

The following environment variables are internal *mailx* variables. They may be imported from the execution environment or set using the set command at any time. The unset command may be used to erase variables.

allnet
All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. The default is **noallnet**. Also see the **alternates** command and the **metoo** environment variable.

append
Upon termination, append messages to the end of the *mbox* file instead of prepending them. The default is **noappend**.

askcc
Prompt for the Cc list after the message is entered. The default is **noaskcc**.

asksub
Prompt for subject if it is not specified on the command line with the −s option. This variable is enabled by default.

autoprint
Enable automatic printing of messages after the **delete** and **undelete** commands. The default is **noautoprint**.

bang
Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). The default is **nobang**.

cmd=*shell-command*
Set the default shell command for the **pipe** command. There is no default value.

conv=*conversion*
Convert *uucp*(1) addresses to the specified address style. The only valid conversion is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. Also see the **sendmail** environment variable and the −U command line option.

crt=*number*
Pipe messages having more than *number* lines through the command specified by the **PAGER** environment variable. The **PAGER** environment variable default is *pg*(1). This **crt** environment variable is disabled by default.

DEAD=*filename*
The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. The default is $HOME/dead.letter.

debug
Enable verbose diagnostics for debugging. Messages are not delivered. The default is **nodebug**.

dot
Take a period on a line by itself during input from a terminal as end-of-file. The default is **nodot**.

EDITOR=*shell-command*
The command to run when the edit or ~e command is used. The default is *ed*(1).

escape=*c*
Substitute *c* for the ~ escape character.

folder=*directory*
The directory for saving standard mail files. User specified

file names beginning with a plus (+) are expanded by preceding the filename with this directory name to obtain the real filename. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a *mailx* command line, the **folder** environment variable must be an exported *sh*(1) environment variable. There is no default for the **folder** environment variable. Also see the **outfolder** environment variable.

header
Enable printing of the header summary when entering *mailx*. This is enabled by default.

hold
Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. The default is **nohold**.

ignore
Ignore interrupts while entering messages. This variable is useful for noisy dial-up lines. The 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. The default is **noignoreeof**. Also see the **dot** environment variable.

keep
When the *mailbox* is empty, truncate it to zero length instead of removing it. This variable is used to preserve the permissions of the *mailbox*. This is disabled by default.

keepsave
Keep messages saved in other files in the *mailbox* instead of deleting them. The default is **nokeepsave**.

MBOX=*filename*
The name of the file to save messages which have been read. The **xit** command overrides this variable as does saving the message explicitly in another file. The default is $HOME/mbox.

metoo
If your login appears as a recipient, do not delete it from the list. The default is **nometoo**.

LISTER=*shell-command*
The command and options to use when listing the contents of the directory specified by the **folder** environment variable. 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 variable 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
Put the files used to record outgoing messages in the directory specified by the **folder** environment variable unless the

pathname is absolute. The default is **nooutfolder**. See the **folder** environment variable and the Save, Copy, followup, and Followup commands.

page
Used with the **pipe** command to insert a form feed after each message sent through the pipe. The default is **nopage**.

PAGER=*shell-command*
Use the shell command as a filter for paginating output. This can also be used to specify the options to be used. The default is *pg*(1).

prompt=*string*
Set the *command mode* prompt to *string*. The default is "? ".

quiet
Do not print the opening message and version when entering *mailx*. The default is **noquiet**.

record=*filename*
Record all outgoing mail in *filename*. This is disabled by default. Also see the **outfolder** environment variable.

save
Enable saving of messages in *dead.letter* on interrupt or delivery error. See the **DEAD** environment variable for a description of this file. This is enabled by default.

screen=*number*
Set the number of lines in a screen-full of headers for the headers command.

sendmail=*shell-command*
Alternate command for delivering messages. The default is *mail*(1).

sendwait
Wait for background mailer to finish before returning. The default is **nosendwait**.

SHELL=*shell-command*
The name of a preferred command interpreter. The 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** command is given. There is no default. Also see the **~i** tilde escape command.

Sign=*string*
The variable inserted into the text of a message when the **~A** command is given. There is no default. Also see the **~i** tilde escape command.

toplines=*number*
The number of lines of header to print with the **top** command. The default is 5.

VISUAL=*shell-command*
The name of a preferred screen editor. The default is *vi*(1).

EXAMPLE

To establish the *mailx* environment at startup, create a file named **.mailrc** in your home directory such as the following.

```
set EDITOR=vi
set keep
set showto
alias thor clarke stone thompson
```

FILES

$HOME/.mailrc	personal startup file
$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	global startup file
/tmp/R[emqsx]*	temporary files

SEE ALSO

mail(1), pg(1), ls(1), uucp(1).

RESTRICTIONS

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 are still evolving.

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.

SUPPORT STATUS

Supported.

NAME

make − maintain, update, and regenerate groups of programs

SYNOPSIS

make [−f makefile] [−p] [−i] [−k] [−s] [−r] [−n] [−b] [−e] [−m] [−t] [−d] [−q] [names]

DESCRIPTION

Make executes commands in *makefile* to update one or more *target* files designated by *names*. *Name* is typically a program. If no −f option is present, **makefile**, **Makefile**, **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is −, the standard input is taken. More than one − *makefile* option pair may appear.

Make updates a target only if that file depends on other newer files (the *prerequistes* for the target). *Make* recursively adds all prerequisite files of a target to the list of targets. *Make* assumes that missing files are out of date.

Makefile contains a sequence of entries that specify *dependencies* (which files depend on other files). The first line of an entry is a blank-separated, non-null list of targets, then a :, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that begin with a tab are shell commands to be executed to update the target. Shell commands may be continued across lines with the <backslash><new-line> sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

produces

```
ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments.

The first line that does not begin with a tab or # begins a new dependency or macro definition.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
     cc a.o b.o −o pgm
a.o: incl.h a.c
     cc −c a.c
b.o: incl.h b.c
     cc −c b.c
```

Make executes command lines one at a time, each by its own shell. The first one or two characters in a command can be the following: −, @, −@, or @−. If @ is present, *make* does not print the command. If − is present, *make* ignores an error.

Make prints each command line when the command is executed unless the −s option is present, or the entry .SILENT: is in

makefile, or the initial character sequence contains a @.

The −n option specifies printing without execution; however, if the command line has the string $(MAKE) in it, the line is always executed (see discussion of the MAKEFLAGS macro under *Environment*).

The −t (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. *Make* ignores errors if the −i option is present, or the entry .IGNORE: appears in *makefile*, or the initial character sequence of the command contains −. . If the −k option is present, *make* abandons work on the current entry, but continues on other branches that do not depend on that entry.

The −b option allows old makefiles (those written for the old version of *make*) to run without errors. The difference between the old version of *make* and this version is that this version requires all dependency lines to have a (possibly null or implicit) command associated with them. The previous version of *make* assumed if no command was specified explicitly that the command was null.

Pressing interrupt or quit deletes the target unless the target is a dependency of the special name .PRECIOUS. (See SPECIAL NAMES below)

OPTIONS

−f *makefile* Assumes *makefile* is the name of a description file. A file name of − denotes the standard input. The contents of *makefile* override the built-in rules if they are present.

−p Print out the complete set of macro definitions and target descriptions.

−i Ignore error codes returned by invoked commands. *Make* also enters this mode if the fake target name .IGNORE appears in the description file.

−k Abandon work on the current entry, but continue on other branches that do not depend on that entry.

−s Enter silent mode, do not print command lines before executing. *Make* also enters this mode if the fake file name .SILENT appears in the description file.

−r Do not use the built-in rules.

−n No execute mode. Print commands, but do not execute them. Note that *make* prints lines beginning with an @ when this option is used.

−b Enter compatibility mode for old makefiles.

−e Override assignments in makefiles with the environment variables.

−m Print a memory map showing text, data, and stack. This option is a no-operation on systems without the *getu* system call.

−t Change the dates on the target files so that they are up-to-date, rather than issuing the usual commands (see *touch*(1)).

−d Enter debug mode. Print out detailed information on files and the times when they were examined.

−q Question. The *make* command returns a zero status code if the file is up-to-date; returns a non-zero status code otherwise.

SPECIAL NAMES

.DEFAULT
Use the commands associated with the name .DEFAULT if it exists, if a file must be made but there are no explicit commands or relevant built-in rules.

.PRECIOUS
Do not remove dependents of this file when quit or interrupt is pressed. Quit and interrupt are signals that are usually generated by the rubout and break keys.

.SILENT
Same effect as the **−s** option.

.IGNORE
Same effect as the **−i** option.

Environment

Make reads the environment; it assumes all variables to be macro definitions and processes them as such. *Make* processes environment variables before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **−e** option overrides the macro assignments in a makefile with the environment.

The MAKEFLAGS environment variable may contain any legal input option (except **−f**, **−p**, and **−d**) defined for the command line. Further, upon invocation, *make* creates MAKEFLAGS if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, MAKEFLAGS always contains the current input options. This proves very useful for *supermakes*. In fact, as noted above, when the **−n** option is used, the command $(MAKE) is executed anyway. Therefore, one can perform a **make −n** recursively on a whole software system to see what would have been executed, because the **−n** is put in MAKEFLAGS and passed to further invocations of $(MAKE). This is one way of debugging all of the makefiles for a software project without actually making changes to any files.

Macros

Entries of the form *string1* = *string2* are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped newline. Subsequent appearances of $(*string1*[:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping

occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

The five internally maintained macros are useful for writing rules for building targets.

$* The macro $* stands for the file name part of the current dependent with the suffix deleted. *Make* evaluates $* only for inference rules.

$@ The $@ macro stands for the full target name of the current target. *Make* evaluates $@ only for explicitly named dependencies.

$< The $< macro (only evaluated for inference rules or the .DEFAULT rule) is the module which is out of date with respect to the target (i.e., the generated dependent file name). Thus, in the .c.o rule, the $< macro would evaluate to the .c file. An example for making optimized .o files from .c files is:

```
.c.o:
        cc -c -O $*.c
```

or:

```
.c.o:
        cc -c -O $<
```

$? The $? macro (evaluated when explicit rules from the makefile are evaluated) is the list of prerequisites that are out of date with respect to the target; essentially, those modules which must be rebuilt.

$% The $% macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, $@ evaluates to **lib** and $% evaluates to the library member, **file.o**.

Each macro, except $? has an alternative form. An upper case **D** or **F** appended to any of the four macros changes the meaning to *directory part* for **D** and *file part* for **F**. Thus, **$(@D)** refers to the directory part of the string $@. If there is no directory part, *./* is generated.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as .c, .s, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, *make* compiles that prerequisite to create the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The default inference rules are:

```
.c .c~ .sh .sh~ .c.o .c~.o .c~.c .s.o .s~.o .y.o .y~.o .l.o .l~.o
.y.c .y~.c .l.c .c.a .c~.a .s~.a .h~.h
```

The source file rules.c contains the internal rules for *make*. These rules can be locally modified. To print out the rules compiled into the *make* in a form suitable for recompilation, the following command is used:

```
make −fp − 2>/dev/null </dev/null
```

The only peculiarity in this output is the (**null**) string which *printf*(3S) prints when handed a null string.

A tilde in the above rules refers to an SCCS file (see *sccsfile*(4)). Thus, the rule .c~.o transforms an SCCS C source file into an object file (.o). Because the s. of the SCCS files is a prefix, it is incompatible with the suffix point-of-view taken by *make*. The tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e. .c:) defines how to build *x* from *x*.c. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for .SUFFIXES. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

```
.SUFFIXES: .o .c .y .l .s
```

Here again, the above command for printing the internal rules displays the list of suffixes implemented. Multiple suffix lists accumulate; .SUFFIXES: with no dependencies clears the list of suffixes.

Inference Rules

The first example can be specified more briefly:

```
pgm: a.o b.o
        cc a.o b.o −o pgm
a.o b.o: incl.h
```

This abbreviation can be made because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

The default inference rules use certain macros to permit the inclusion of optional matter in any resulting commands. For example, CFLAGS, LFLAGS, and YFLAGS are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1) respectively. Again, the command suggested for examining the current rules (see *suffixes*) is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix .o from a file with suffix .c is specified as an entry with .c.o: as the target and no dependents. Shell commands associated with the target define the rule for making a .o file from a .c file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, *make*

assumes that the file is an archive library, the string within parentheses referring to a member within the library. For example, **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library which contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression

$(LIB)(file1.o file2.o)

is not legal. Rules pertaining to archive libraries have the form *.XX*.**a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up to date
.c.a:
        $(CC) -c $(CFLAGS) $<
        ar rv $@ $*.o
        rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        ar rv lib $?
        rm $?  @echo lib is now up to date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is the set of object file names (inside **lib**) whose C source files are out of date. The substitution mode translates the **.o** to **.c**. (One cannot transform to **.c~**.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

Release 2.x archive formats are supported permitting transparent usage with archive libraries from Release 2.x.

FILES

[Mm]akefile and s.[Mm]akefile

SEE ALSO

cc(1), cd(1). lex(1), sh(1), yacc(1).
A Program for Maintaining Computer Programs (make) and *Augmented Version of make* in the Support Tools Guide.

RESTRICTIONS

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty.

Filesnames with the characters =, :, or @ do not work.

Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across new-lines in *make*.

The syntax **(lib(file1.o file2.o file3.o)** is not valid. You cannot build **lib(file.o)** from **file.o.**

The macro **$(a:.o=.c~)** does not work.

SUPPORT STATUS

Supported.

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

Makekey is intended for programs that perform encryption (e.g., *ed*(1) and *crypt*(1)). Usually, its input and output are pipes.

SEE ALSO

crypt(1), ed(1), passwd(4).

SUPPORT STATUS

Supported.

NAME

man − print entries in this manual

SYNOPSIS

man [options] [section] titles

DESCRIPTION

Man is an online help system and is available if your system administrator installed the manual files on your system. *Man* locates and prints an entry of this manual, the *Programmer Reference Manual* or the *Superuser Reference Manual* named *title* in the specified *section*. The *title* is entered in lower case. The *section* number may not have a letter suffix. If no *section* is specified, all manuals are searched for *title* and all occurrences of it are printed. *Section* may be changed before each *title*.

OPTIONS

Man examines the environment variable **$TERM** (see *environ*(5)) and attempts to select options that adapt the output to the terminal being used. The −T*term* option overrides the value of **$TERM**; in particular, one should use **−Tlpr** when sending the output of *man* to a line printer.

−T*term*
: Print the entry as appropriate for terminal type *term*. For a list of the recognized values of *term*, type **help term2**. The default value of *term* is **450**.

−w
: Print on the standard output only the *path names* of the entries, relative to **/usr/catman**, or to the current directory for **−d** option.

−d
: Search the current directory rather than **/usr/catman**; requires the full file name (e.g., **cu.1c**, rather than just **cu**).

−c
: Invoke *col*(1); note that *col*(1) is invoked automatically by *man* unless *term* is one of **300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620,** or **X**.

EXAMPLE

To display the entry for the *man* command, enter

```
man 1 man
```

FILES

/usr/catman/?_man/man[1-8]/* preformatted manual entries

SUPPORT STATUS

Supported.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg permits or denys messages to be received by your terminal from another user via *write*(1).

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the terminal of the user. *Mesg* with argument **y** reinstates permission. *Mesg* with no argument reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

The exit status is 0 is messages are permitted, 1 if they are denied, or 2 if an error occurred.

SUPPORT STATUS

Supported.

NAME

mkdir – make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories. Standard entries .(dot) for the directory itself and ..(dot**dot**) for its parent are made automatically.

Mkdir typically creates directories in mode 777 which are readable, writable, and searchable by the owner, group, and everyone. *Mkdir* requires write permission in the parent directory.

EXAMPLE

To make *may*, *june*, and *july* directories in the current directory enter

```
mkdir may june july
```

SEE ALSO

sh(1), rm(1), umask(1).

DIAGNOSTICS

Exit code is 0 if all directories are successfully made or non-zero if and error occurred.

SUPPORT STATUS

Supported.

NAME

mklost+found − make a lost+found directory for fsck

SYNOPSIS

/etc/mklost+found

DESCRIPTION

A directory *lost+found* is created in the current directory and a number of empty files are created therein and then removed so that there are empty slots for *fsck*(1).

This command should be run immediately after first mounting and changing directory to a newly created file system.

For small file systems, it is sufficient (and much faster) to simply make a lost+found directory. Up to 30 files can be recovered in it.

SEE ALSO

fsck(1)

SUPPORT STATUS

Supported.

NAME

mkstr − create an error message file by massaging C source

SYNOPSIS

mkstr [−] messagefile prefix file ...

DESCRIPTION

Mkstr creates files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program because the error messages do not have to be constantly swapped in and out.

Mkstr processes each of the specified *files*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. A typical usage of *mkstr* would be

```
mkstr pistrings xx *.c
```

This command would place all the error messages from the C source files in the current directory into the file *pistrings* and place processed copies of the source for these files into files whose names are prefixed with *xx*.

To process the error messages in the source to the message file *mkstr* keys on the string

```
error("
```

in the input stream. Each time it occurs, the C string starting at the double quote is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The altered copy of the input file then contains a *lseek* pointer into the file which can be used to retrieve the message, i.e.:

```
char    efilname[] =  "/usr/lib/pi_strings";
int     efil = -1;

error(a1, a2, a3, a4)
{
        char buf[256];

        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
                goto oops;
        printf(buf, a2, a3, a4);
}
```

OPTIONS

The optional – places the error messages at the end of the specified message file for recompiling part of a large *mkstr* ed program.

SEE ALSO

lseek(2), xstr(1)

SUPPORT STATUS

Supported.

NAME

mm, osdd, checkmm − print/check documents formatted with the MM macros

SYNOPSIS

mm [options] [files]

osdd [options] [files]

checkmm [files]

DESCRIPTION

Mm can be used to type out documents using *nroff* and the MM text-formatting macro package. It has options to specify preprocessing by *tbl*(1) and/or *neqn* (see *eqn*(1)) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff* and MM are generated, depending on the options selected.

Mm reads the standard input when − is specified instead of any file names. (Mentioning other files together with − leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

cat report | mm −

Osdd is equivalent to the command **mm −mosd**. For more information about the OSDD adapter macro package, see *mosd*(5).

Checkmm is a program for checking the contents of the named *files* for errors in the use of the Memorandum Macros, missing or unbalanced *neqn* delimiters, and **.EQ/.EN** pairs. Note: The user need not use the *checkeq* program (see *eqn*(1)). Appropriate messages are produced. The program skips all directories, and if no file name is given, standard input is read.

OPTIONS

Any other arguments or options (e.g., **−rC3**) other than those below are passed to *nroff* or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

−T*term* Specifies the type of output terminal; for a list of recognized values for *term*, type **help term2**. If this option is *not* used, *mm* uses the value of the shell variable **$TERM** from the environment (see *profile*(4) and *environ*(5)) as the value of *term*, if **$TERM** is set; otherwise, *mm* uses **450** as the value of *term*. If several terminal types are specified, the last one takes precedence.

−12 Produces the document in 12-pitch. May be used when **$TERM** is set to one of **300**, **300s**, **450**, and **1620**. (The pitch switch on the DASI 300 and 300s terminals must be manually set to 12 if this option is used.)

−c Invokes *col*(1); note that *col*(1) is invoked automatically by *mm* unless *term* is one of **300**, **300s**, **450**, **37**, **4000a**, **382**, **4014**, **tek**, **1620**, and **X**.

−e Invokes *neqn*; also causes *neqn* to read the **/usr/pub/eqnchar** file (see *eqnchar*(5)).

−t Invokes *tbl*(1).

−E Invokes the −e option of *nroff*.

−y Uses the non-compacted version of the macros (see *mm*(5)).

EXAMPLE

Assuming that the shell variable $TERM is set in the environment to 450, the two command lines below are equivalent:

```
mm −t −rC3 −12 chapter*
tbl chapter* | nroff −cm −T450−12 −h −rC3
```

HINTS

1. *Mm* invokes *nroff* with the −h option. With this option, *nroff* assumes that the terminal has tabs set every 8 character positions.
2. Use the −o*list* option of *nroff* to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of the −e, −t, and − options, *together* with the −o*list* option of *nroff* may cause a harmless *broken pipe* diagnostic if the last page of the document is not specified in *list*.
3. If you use the −s option of *nroff* (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The −s option of *nroff* does not work with the −c option of *mm*, or if *mm* automatically invokes *col*(1) (see −c option above).
4. If you mistake the kind of terminal the output from *mm* will be printed on, you get (often subtle) garbage; however, if you are redirecting output into a file, use the −T37 option, and then use the appropriate terminal filter when you actually print that file.

SEE ALSO

col(1), env(1), eqn(1), greek(1), mmt(1), nroff(1), tbl(1), profile(4), mm(5), mosd(5), term(5).

DIAGNOSTICS

mm *mm: no input file*
No arguments are readable files and *mm* is not used as a filter.

checkmm *Cannot open filename*
Unreadable file(s). The remaining output of the program is diagnostic of the source file.

SUPPORT STATUS

Supported.

NAME

mmt, mvt − typeset documents, viewgraphs, and slides

SYNOPSIS

mmt [options] [files]

mvt [options] [files]

DESCRIPTION

Mmt and *mvt* commands are very similar to *mm*(1), except that they both typeset their input via *troff*(1), as opposed to formatting it via *nroff*(1). *Mmt* uses the MM macro package, while *mvt* uses the Macro Package for View Graphs and Slides. These two commands have options to specify preprocessing by *tbl*(1) and/or *pic*(1) and/or *eqn*(1). The proper pipelines and the required arguments and options for *troff*(1) and for the macro packages are generated, depending on the options selected.

These commands read the standard input when − is specified instead of any file names.

Mvt is just a link to *mmt*.

OPTIONS

Options are given below. Any other arguments or options (e.g., **−rC3**) are passed to *troff*(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, these commands print a list of their options.

−e Invoke *eqn*(1) and cause *eqn* to read the **/usr/pub/eqnchar** file (see *eqnchar*(5)).

−t Invoke *tbl*(1).

−p Invoke *pic*(1).

−Taps
Create output for an Autologic APS-5 phototypesetter and send it to the default destination at this installation.

−T*dest*
Create output for *troff* device *dest* (see *troff*(1)). The output is sent through the appropriate postprocessor (see *daps*(1)).

−Tcat
Use *otroff*(1) to generate output for an on-line Wang CAT phototypesetter.

−D4014
Direct the output to a TEKTRONIX 4014 terminal via the *tc*(1) filter.

−Dtek
Same as **−D4014**.

−Dil0
Direct the output to the local Imagen Imprint-10 laser printer.

−a Invoke the **−a** option of *troff*(1).

−y Cause *mmt* to use the non-compacted version of the macros. This is the default except when using −Tcat.

−z Invoke no output filter to process or redirect the output of *troff*(1).

HINT

Use the −o*list* option of *troff*(1) to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the −e, −t, and − options, *together* with the −o*list* option of *troff*(1) may cause a harmless broken pipe diagnostic if the last page of the document is not specified in *list*.

SEE ALSO

daps(1), env(1), eqn(1), mm(1), nroff(1), pic(1), tbl(1), tc(1), profile(4), environ(5), mm(5), mv(5).

DIAGNOSTICS

"m[mv]t: no input file" if none of the arguments is a readable file and the command is not used as a filter.

SUPPORT STATUS

Supported.

NAME

more, page − file perusal filter for crt viewing

SYNOPSIS

more [**−cdflsu**] [*−n*] [*+linenumber*] [*+/pattern*] [name ...]

page *more options*

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a soft-copy terminal.

More normally pauses after each screenful, printing --More-- at the bottom of the screen. If the user then presses a carriage return, *more* displays one more line. If the user presses the space bar, *more* displays another screenful.

Other possibile responses are enumerated in COMMAND CHARACTERS.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

If the standard output is not a teletype, then *more* acts just like *cat*, except that a header is printed before each file (if there is more than one).

Page.

If the program is invoked as *page*, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and $k - 1$ rather than $k - 2$ lines are printed in each screenful, where k is the number of lines the terminal can display.

Window size.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

Environment.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the *−c* mode of operation, or the *sh* command sequence *MORE='-c' ; export MORE* would cause all invocations of *more*, including invocations by programs such as *man* and *msgs*, to use this mode. Normally, the user places the command sequence which sets up the *MORE* environment variable in the *.profile* file.

COMMAND CHARACTERS

Other sequences which may be typed when *more* pauses, and their effects, are described below (*i* is an optional integer argument, defaulting to 1).

The commands take effect immediately; that is, it is not necessary to press a carriage return. Up to the time when the command character itself is given, the user may press the line kill character to cancel the numerical argument being formed. In addition, the user may press the erase character to redisplay the --More--(xx%) message.

i<space> Display *i* more lines, (or another screenful if no argument is given).

^D Display 11 more lines (a *scroll*). If *i* is given, then the scroll size is set to *i*.

d Same as ^D (control-D).

*i*z Same as typing a space except that *i*, if present, becomes the new window size.

*i*s Skip *i* lines and print a screenful of lines.

*i*f Skip *i* screenfuls and print a screenful of lines.

q Exit from *more*.

Q Exit from *more*.

= Display the current line number.

v Start up the editor *vi* at the current line.

h Help command; give a description of all the *more* commands.

i/expr Search for the *i*-th occurrence of the regular expression *expr*.

If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found.

The user may use erase and kill characters to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n Search for the *i*-th occurrence of the last regular expression entered.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!*cmd* Invoke a shell with the command *cmd*. The characters % and ! in *cmd* are replaced with the current file name and the previous shell command respectively. If there is no current file name, % is not expanded. The sequences % and ! are replaced by % and ! respectively.

i:n Skip to the *i*-th next file given in the command line (skips to last file if *n* is not sensible).

i:p Skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f Display the current file name and line number.

:q Same as q.

:Q Same as Q.

. (dot) Repeat the previous command.

At any time when output is being sent to the terminal, the user can press the quit key (normally control−\). *More* stops sending output, and displays the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, because any characters waiting in the output queue of the terminal are flushed when the quit signal occurs.

OPTIONS

The command line options are:

−*n* Use a window *n* lines long (where *n* is an integer) instead of the default window size.

−c Print each page by beginning at the top of the screen and erasing each line just before printing over it. This avoids scrolling the screen, making it easier to read while *more* is writing.

This option will be ignored if the terminal does not have the ability to clear to the end of a line.

−d Prompt the user with the message

Hit space to continue, rubout to abort

at the end of each screenful.

−f Count logical, rather than screen lines. That is, do not fold long lines.

This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.

−l Do not treat ^L (form feed) specially.

If this option is not given, *more* pauses after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

−s Squeeze multiple blank lines from the output, producing only one blank line.

Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

−u Do not attempt to underline on the terminal.

Normally, *more* handles underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if

the terminal can perform underlining or has a stand-out mode, *more* generates appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file.

+*linenumber*
Start at *linenumber.*

+/*pattern*
Start two lines before the line containing the regular expression *pattern.*

EXAMPLE

A sample usage of *more* in previewing *nroff* output would be

```
nroff −ms +2 doc.n | more -s
```

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

SEE ALSO

man(1), sh(1), environ(5)

RESTRICTIONS

When performing *more*(1B), the user may **not** redirect *stderr* to any terminal *(/dev/tty)*. To do so causes *more* to abort after displaying the first screen.

SUPPORT STATUS

Supported.

NAME

newform − change the format of a text file

SYNOPSIS

newform [−s] [−itabspec] [−otabspec] [−bn] [−en] [−pn] [−an] [−f] [−cchar] [−ln] [files]

DESCRIPTION

Newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

OPTIONS

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. For example, −e15 −l60 yields results different from −l60 -e15. Options are applied to all *files* on the command line.

−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.

Newform does not prompt the user if a *tabspec* is to be read from the standard input (by use of −i−− or −o−−).

−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 are converted to tabs on output.

−l*n* Sets the effective line length to *n* characters. If *n* is not entered, −l defaults to 72. If −l is not specified, the line length is assumed to be 80 characters. Tabs and backspaces are considered to be one character (use −i to expand tabs to spaces).

−ln must be used in conjunction with and precede one of the following options:

−bn or −en if the effective line length is less than the existing line length.

−pn or −an if the effective line length is greater than the existing line length.

−b*n* Truncates *n* characters from the beginning of the line when the line length is greater than the effective line length (see −l*n*). If **-b** is not specified, or if **n** is omitted, *newform* truncates the number of characters necessary to obtain the effective line length.

−e*n* Same as −b*n* except that characters are truncated from the end of the line.

−c*k* Changes the prefix/append character to *k*. Default character for *k* is a space. (See **-pn.**)

−p*n* Prefixes *n* characters (see −c*k*) to the beginning of a line when the line length is less than the effective line length. If **-p** is not specified, *newform* prefixes 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. (See also **-c***k*.)

−f Writes the tab specification format line on the standard output before any other lines are output. The tab specification format line corresponds to the format specified in the *last* −o option. If no −o option has been specified, the tab specification format line contains the default specification of −8.

−s Removes leading characters on each line up to the first tab and places up to 8 of the removed characters at the end of the line. If more than 8 characters (not counting the first tab) are removed, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

The characters removed 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. An error message and program exit occurs if this option is used on a file without a tab on each line.

EXAMPLES

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
```

The −b option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform −l1 −b7 file-name
```

The −l1 must be used to set the effective line length shorter than any existing line in the file so that the −b option is activated.

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*Newform* was called with a bad option.
not −s format	There was no tab on one line.
cannot 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).

EXIT CODES

0 − normal execution
1 − for any error

SEE ALSO

csplit(1), tabs(1), fspec(4).

RESTRICTIONS

Newform normally only keeps track of physical characters; however, for the −i and −o options, *newform* keeps track of backspaces in order to line up tabs in the appropriate logical columns.

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 is incorrect.

SUPPORT STATUS

Supported.

NAME

newgrp − log in to a new group

SYNOPSIS

newgrp [−] [group]

DESCRIPTION

Newgrp 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** is now 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.

If the first argument to *newgrp* is a −, the environment is changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in /etc/**group** as being a member of that group.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), sh(1), group(4), passwd(4), environ(5).

RESTRICTIONS

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.

SUPPORT STATUS

Supported.

NAME

news − print news items

SYNOPSIS

news [−a] [−n] [−s] [items]

DESCRIPTION

News is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**. When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *Items* are names of specific news items that are to be printed.

If a *delete* is pressed 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.

News stores the *current* time as the modification date of a file named **$HOME/.news_time**. Only files more recent than this current time are considered *current*.

OPTIONS

If any of the options below are used, *news* does not change the stored time.

−a Print all items regardless of *current* time.
−n Report only the names of the *current* items.
−s Report only how many *current* items exist.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(4), environ(5).

SUPPORT STATUS

Supported.

NAME

nice − run a command at low priority

SYNOPSIS

nice [−increment] command [arguments]

DESCRIPTION

Nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

An *increment* larger than 19 is equivalent to 19.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., **−−10**.

SEE ALSO

nohup(1), nice(2).

DIAGNOSTICS

Nice returns the exit status of the subject command.

SUPPORT STATUS

Supported.

NAME

nl − line numbering filter

SYNOPSIS

nl [−**h**type] [−**b**type] [−**f**type] [−vstart#] [−iincr] [−**p**] [−lnum] [−ssep] [−wwidth] [−nformat] [−ddelim] file

DESCRIPTION

Nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

Nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g. no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

OPTIONS

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* Number the logical page body lines according to *type*. Recognized *types* and their meaning are:

- **a** number all lines
- **t** number lines with printable text only
- **n** no line numbering
- p*string* number only lines that match the regular expression *string*

Type for logical page body is assumed to be **t**.

−**h***type* Number the logical page header according to *type* (see **-b**). Default *type* is **n**.

−**f***type* Number the logical page footer according to *type*. (See **-b**). Default *type* is **n**.

−**p** Do not restart numbering at logical page delimiters.

−v*start#* Number logical page lines with *start#* as the initial value. Default *start#* is 1.

−i*incr* Number logical page lines with *incr* as the increment value. Default *incr* is 1.

−s*sep* Separate the line number and the corresponding text line with the chararacter *sep*. Default *sep* is a tab.

−w*width* Use *width* number of characters for the line number. Default *width* is 6.

−n*format*
Use *format* as the line numbering format. Recognized values are:

ln left justified, leading zeroes supressed
rn right justified, leading zeroes supressed
rz right justified, leading zeroes kept.

Default *format* is **rn** (right justified).

−l*num* Consider *num* blank lines 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 *num* is 1.

−d*xx* Change the delimiter characters for the start of a logical page section 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
```

numbers file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

SUPPORT STATUS

Supported.

NAME

nm − print name list of common object file

SYNOPSIS

nm [−o] [−x] [−h] [−v] [−n] [−e] [−f] [−u] [−V] [−T] *filenames*

DESCRIPTION

The *nm* command displays the symbol table of each common object file *filename*. *Filename* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information is printed:

Name The name of the symbol.

Value Its value expressed as an offset or an address depending on its storage class.

Class Its storage class.

Type Its type and derived type. If the symbol is an instance of a structure a of a union, the structure or union tag is given following the type (e.g., struct-tag). If the symbol is an array, the array dimensions are given following the type (e.g., **char**[n][m]). Note that the object file must have been compiled with the **−g** option of *cc*(1) for this information to be output.

Size Its size in bytes, if available. Note that the object file must have been compiled with the **−g** option of the *cc*(1) command for this information to be output.

Line The source line number at which it is defined, if available. Note that the object file must have been compiled with the **−g** option of the *cc*(1) command for this information to be output.

Section For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).

OPTIONS

The output of *nm* may be controlled using the following options. Options may be used in any order, either singly or in combination, and may appear anywhere in the command line.

−o Print the value and size of a symbol in octal instead of decimal.

−x Print the value and size of a symbol in hexadecimal instead of decimal.

−h Suppress the output header data.

−v Sort external symbols by value before printing them.

−n Sort external symbols by name before printing them.

−e Print only static and external symbols.

−f Produce full output, including redundant symbols (.text, .data and .bss) normally suppressed.

−u Print only undefined symbols.

−V Display the version of *nm* command executing on standard error output.

−T By default, *nm* prints the entire name of the symbols listed. Because object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names overflows its column, forcing every column after the name to be misaligned. The **−T** option causes *nm* to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

EXAMPLE

This command prints the static and external symbols in **file**, with external symbols sorted by value:

```
nm file −e −v
```

This command does the same:

```
nm −ve file
```

FILES

/usr/tmp/nm??????

WARNINGS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve the scoping information. Therefore, the **−v** and **−n** options should be used only in conjunction with the **−e** option.

SEE ALSO

as(1), cc(1), ld(1), a.out(4), ar(4).

DIAGNOSTICS

nm: name: cannot open
Name cannot be read.

nm: name: bad magic
Name is not an appropriate common object file.

nm: name: no symbols
The symbols have been stripped from *name*.

SUPPORT STATUS

Supported.

NAME

nohup − run a command immune to hangups and quits

SYNOPSIS

nohup command [arguments]

DESCRIPTION

Nohup executes *command* with hangups and quits ignored.

If you log off (hangup) while a command is executing in the background, the command terminates. Using *nohup* to execute a background command causes the command to continue execution if you log off.

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:

```
tbl ofile | eqn | nroff > nfile
```

In the following command format, *nohup* applies only to command 1.

```
nohup command1; command2
```

The following command format is syntactically incorrect.

```
nohup (command1; command2)
```

If output is not redirected 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**. Be careful of where standard error is redirected. The following command may put error messages on tape making it unreadable.

```
nohup cpio −o <list >/dev/rmt/0yy&
```

This command puts the error messages into file *errors*.

```
nohup cpio −o <list >/dev/rmt/0yy 2>errors&
```

SEE ALSO

chmod(1), nice(1), sh(1), signal(2).

SUPPORT STATUS

Supported.

NAME

nroff, troff − format or typeset text

SYNOPSIS

nroff [options] [files]

troff [options] [files]

DESCRIPTION

Nroff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers; similarly, *troff* formats text for a Wang Laboratories, Inc., C/A/T phototypesetter.

OPTIONS

An argument consisting of a minus (−) is taken to be a file name corresponding to the standard input. The *options*, which may appear in any order, but must appear before the *files*, are:

−o*list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N*−*M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end. (See *RESTRICTIONS* below.)

−n*N* Number first generated page *N*.

−s*N* Stop every *N* pages. *Nroff* halts *after* every *N* pages (default *N*=1) to allow paper loading or changing, and resumes upon receipt of a line-feed or new-line (new-lines do not work in pipelines, e.g., with *mm*(1)). This option does not work if the output of *nroff* is piped through *col*(1). *Troff* stops the phototypesetter every *N* pages, produces a trailer to allow changing cassettes, and resumes when the typesetter start button is pressed. When *nroff* (*troff*) halts between pages, an ASCII BEL (in *troff*, the message **page stop**) is sent to the terminal.

−r*aN* Set register *a* (which must have a one-character name) to *N*.

−**i** Read standard input after *files* are exhausted.

−**q** Invoke the simultaneous input-output mode of the **.rd** request.

−**z** Print only messages generated by **.tm** (terminal message) requests.

−m*name* Prepend to the input *files* the non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.

−c*name* Prepend to the input *files* the compacted macro files **/usr/lib/macros/cmp.[nt].[dt].***name* and **/usr/lib/macros/ucmp.[nt].***name*.

−k*name* Compact the macros used in this invocation of *nroff*/*troff*, placing the output in files **[dt].***name* in the current directory.

Nroff only:

−T*name* Prepare output for specified terminal. Known *name*s are **37** for the (default) TELETYPE® Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300s** for the DASI 300s, **300** for the DASI 300, **450** for the DASI 450, **lp** for a (generic) ASCII line printer, **382** for the DTC-382, **4000A** for the Trendata

4000A, 832 for the Anderson Jacobson 832, X for a (generic) EBCDIC printer, 2631 for the Hewlett Packard 2631 line printer, 6411 for the NCR 6411 printer, 6416 for the NCR 6416 printer, and 6455 for the NCR 6455 printer.

−e Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.

−h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

−u*n* Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

Troff only:

−t Direct output to the standard output instead of the phototypesetter.

−f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

−w Wait until phototypesetter is available, if it is currently busy.

−b Report whether the phototypesetter is busy or available. No text processing is done.

−a Send a printable ASCII approximation of the results to the standard output.

−p*N* Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

−Tcat Use font-width tables for Wang CAT phototypesetter. This device is both the default and the only choice.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*
/usr/lib/font/*	font width tables for *troff*

SEE ALSO

col(1), eqn(1), greek(1), mm(1), mmt(1), tbl(1), troff(1), mm(5).

RESTRICTIONS

Nroff/troff internally supports Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *nroff/troff* generates may be off by one day from your idea of what the date is.

When *nroff/troff* is used with the −*olist* option inside a pipeline (e.g., with *eqn*(1), or *tbl*(1)), it may cause a harmless broken pipe diagnostic if the last page of the document is not specified in *list*.

SUPPORT STATUS

Supported.

NAME

od − octal dump

SYNOPSIS

od [−bcdosx] [file] [[+]offset[.][b]]

DESCRIPTION

Od dumps *file* in one or more formats as selected by the first argument. If no options are specified, **−o** is assumed.

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 octal bytes) in the file where dumping is to commence. 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.

OPTIONS

−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.

SEE ALSO

dump(1).

SUPPORT STATUS

Supported.

NAME

pack − compress files

SYNOPSIS

pack [−] [−f] name ...

pcat name ...

unpack name ...

DESCRIPTION

Pack attempts to compress the specified files. Packed files can be restored to their original form using *unpack*(1) or *pcat*(1).

Wherever possible, each input file *name* is replaced by a packed file *name*.z with the same access modes, access and modified dates, and owner as those of *name*. If *pack* is successful, *name* is removed. The −f option forces packing of *name*. This is useful for causing an entire directory to be packed even if some files do not benefit.

If the − argument is used, an internal flag is set which causes the number of times each byte is used, its relative frequency, and the code for the byte to be output on the standard output. Additional occurrences of − in place of *name* set and reset the internal flag.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. 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, *pack* reduces text files to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

Pack returns the number of files that it failed to compress.

No packing occurs 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 are saved by packing;
a file called *name*.z already exists;
the .z file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(1) does for ordinary files, except that *pcat* can not 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

Pcat returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the .z) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of *pack*.

Unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

a file with the unpacked name already exists;
if the unpacked file cannot be created.

SEE ALSO
cat(1).

SUPPORT STATUS
Supported.

NAME

packsf, unpacksf − compress and uncompress sparse file

SYNOPSIS

packsf <input_file >compressed_file
unpacksf <compressed_file >original_file

DESCRIPTION

Packsf compresses a sparse file, a file that has a large ratio of zeros to nonzero data, into a formatted file that takes less space. The compressed file can be used to recreate the original file.

If the original file is mostly nonzero, other utilities provide better compression.

The compressed file is an array of variable length records. The format of the records is:

```
/* Maximum size is 1024 */
struct record {
        int r_faddr;     /* File address */
        int r_len;       /* Length of the data */
        unsigned char r_buffer[1016];
};
```

Each record represents a region of nonzero data in the original file. *R_faddr* holds the file offset, from 0, of the nonzero region and *r_len* gives the length of the region. *R_buffer* holds the data.

If two nonzero regions are seperated by less than 9 zeros the zeros are not compressed.

Unpacksf reverses the compression and restores the original file. *Unpacksf* reads records from the compressed file and executes them as commands of the form:

SEEK TO r.r_faddr AND WRITE r.r_len BYTES FROM r.r_buffer

SEE ALSO

pack(1).

SUPPORT STATUS

Supported.

NAME

passwd – change login password

SYNOPSIS

passwd [name]

DESCRIPTION

The *passwd* command changes or installs a password associated with the login *name*. Ordinary users may change only the password which corresponds to their login *name*.

Passwd prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered *passwd* checks to see if the old password has aged sufficiently. If aging is insufficient the new password is rejected and *passwd* terminates; see *passwd*(4).

Assuming aging is sufficient, a check is made to insure 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 means upper and lower case 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 upper case 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 upper case letter and its corresponding lower case letter are equivalent.

One whose effective user identification is zero is called a superuser or root. Superusers may change any password; hence, *passwd* does not prompt superusers for the old password. Superusers are not forced to comply with password aging and password construction requirements. A superuser can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd

SEE ALSO

login(1), id(1), su(1), crypt(3C), passwd(4).

SUPPORT STATUS

Supported.

NAME

paste − merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste −**d**list file1 file2 ...
paste −**s** [−**d**list] file1 file2 ...

DESCRIPTION

In the first two command 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). *Paste* is the horizontal counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other.

In the last command form above, *paste* combines subsequent lines of the input file (serial merging).

In all cases, lines are connected with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so *paste* can be used as the start of a pipe, or as a filter, if − is used in place of a file name.

OPTIONS

-d*list*

Replace the *tab* character by one or more alternate characters specified in *list*. 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
\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* "\\\\").

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.

−**s** Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless the −**d** option is used. The very last character of the file is a new-line.

− May be used in place of any file name, to read a line from the standard input. *Paste* does not prompt.

NOTE

pr −**t** −**m**... works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

EXAMPLES

ls | paste −d" " − list directory in one column

ls | paste − − − −	list directory in four columns
paste −s −d"\t\n" file	combine pairs of lines into lines

SEE ALSO

grep(1), cut(1), pr(1).

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.

SUPPORT STATUS

Supported.

NAME

pcdsk – PC-DOS to UNIX file transfer

SYNOPSIS

pcdsk

DESCRIPTION

Pcdsk transfers files between a PC-DOS (or MS-DOS) floppy disk and the UNIX file system and provides directory listing functions.

Pcdsk handles 5.25 inch floppy disks formatted for PC-DOS version 2.1 which are single or double sided, have eight or nine sectors per track, and are formatted 48 tracks per inch.

The PC-DOS floppy disk must be installed in the top, left, or only floppy disk drive. The superuser must make a special file (see *wd*(7)) using *mknod*(1M) before *pcdsk* can be executed.

To specify a pathname for a *pcdsk* copy operation, use a UNIX-style pathname although either / or \ may be used to separate pathname parts. The metacharacters * and ? are recognized to have their UNIX meaning in pathnames. No escape character such as \ is recognized.

COMMANDS

cat List a UNIX file. This command is identical to the UNIX *cat*(1) command; all *cat* options are accepted. *Pcdsk* passes this command to the shell for execution.

dir List the directory of the PC-DOS floppy disk.

exit Terminate *pcdsk*.

help Display the *pcdsk* commands and command descriptions.

ls List a UNIX directory. This command is identical to the UNIX *ls*(1) command; all *ls* options are accepted. *Pcdsk* passes this command to the shell for execution.

mtu Copy files from the PC-DOS floppy disk to the UNIX file system. After this command is entered, *pcdsk* prompts for the PC-DOS source pathname and the UNIX destination pathname. Metacharacters (wildcards) may be entered in the PC-DOS pathname, but may not be entered in the UNIX destination pathname.

sh Invoke the UNIX shell. To exit the shell and return to *pcdsk*, enter a control-d.

utm Copy files from the UNIX file system to the PC-DOS floppy disk. After this command is entered, *pcdsk* prompts for the UNIX source pathname and the PC-DOS destination pathname. Metacharacters (wildcards) may be entered in the UNIX pathname, but may not be entered in the PC-DOS destination pathname.

!*command*

Escape to the shell and execute *command*.

control-d

Terminate *pcdsk*. This is the same as the exit command.

EXAMPLES

If the PC-DOS source pathname and the UNIX destination pathname in a copy operation are specified as:

From pcdos files: /PCSUBDIR/PCFILE
To UNIX files: usubdir/ufile

the **PCFILE** file is copied to the **ufile** file. The PC-DOS pathname specification is from the PC-DOS root directory. The UNIX pathname specification is from the current working directory.

If the PC-DOS source pathname and the UNIX destination pathname in a copy operation are specified as:

From pcdos files: PCSUBDIR/PCFILE
To UNIX files: /usr/acct/thompson/usubdir/ufile

the **PCFILE** file is copied to the **ufile** file. The PC-DOS pathname specification is from the PC-DOS root directory even though a / is not specified. The UNIX pathname specification is from the root directory because / is the first character specified in the pathname.

If the PC-DOS source pathname and the UNIX destination pathname in a copy operation are specified as:

From pcdos files: /PCSUB*/PCFILE.??
To UNIX files: usubdir

then starting at the PC-DOS root directory, all files which are named **PCFILE.** with any two character file name extention in any directory which has a name starting with **PCSUB** are copied to the **usubdir** directory which is a subdirectory of the current working directory.

SEE ALSO

mknod(1M), wd(7).

RESTRICTIONS

When a file is copied to a PC-DOS floppy disk, the date and time are not put in the directory entry.

The [and] metacharacters do not work.

Pcdsk does not create a PC-DOS floppy disk directory nor does it format a floppy disk.

SUPPORT STATUS

Not supported.

NAME

pg − file perusal filter for screen terminals

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 terminal. The file name − or no file name indicates 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.

OPTIONS

The command line options are:

−*number*
: The number of lines in the window that *pg* is to use instead of the default number. On a terminal containing 24 lines, the default window size is 23.

−**p** *string*
: Use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt is replaced by the current page number when the prompt is issued. The default prompt string is ":".

−**c** Clear the screen and home the cursor 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** Do not pause at the end of each file.

−**f** Do not split lines. 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.

−**n** Cause an automatic end of command as soon as a command letter is entered. Normally, commands must be terminated by a newline.

−**s** Print all messages and prompts in standout mode (usually inverse video).

+*linenumber*
: Start up at *linenumber*.

+/*pattern*/
: Start up at the first line containing the regular expression pattern.

COMMANDS

At any time the prompt is displayed, the user may enter one of the *pg* commands. When output is being sent to the terminal, the user can press the quit key (normally control-\) or the interrupt (break) key to cause *pg* to stop sending output and display the prompt. Some output is lost when this is done because characters in the

terminal output queue are flushed when the quit signal occurs.

The commands that may be entered when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Further Perusal Commands

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:

(+1)<*newline*> or <*blank*>
: Display one page. The address is specified in pages.

(+1) **l**
: Scroll the screen, forward or backward, the number of lines specified if the *address* is signed. Print a screenful beginning at the specified line if the *address* is unsigned.

(+1) **d** or **^D**
: Scroll half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L**
: Redisplay the current page of text.

$
: Display the last windowful in the file. Use with caution when the input is a pipe.

Search Commands

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 do not properly handle the ?.

After searching, *pg* normally displays 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.

Modify Environment Commands

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 with a default value of 1.

*i***p** Begin perusing the *i*th previous file in the command line. The *i* is an unsigned number with a default value of 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.

EXAMPLE

A sample usage of *pg* in reading system news is:

```
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 because an interrupt is likely to terminate the other commands in the pipeline.

Users of *more*(1) will find that the **z** and **f** commands are available, and that the terminal /, **^**, or ? may be omitted from the searching commands.

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.

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).

FILES

/usr/lib/terminfo/*	terminal information data base
/tmp/pg*	temporary file if pipe input

SEE ALSO

crypt(1), ed(1), grep(1), more(1), pr(1), terminfo(4).

RESTRICTIONS

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 (e.g., *crypt*(1)), terminal settings may not be restored correctly.

SUPPORT STATUS

Supported.

NAME

pr − print files

SYNOPSIS

pr [options] [files]

DESCRIPTION

Pr prints the named files separating the listing into pages. Each page is headed by the page number, a date and time, and the name of the file, unless options specify otherwise. *Pr* is typically used to paginate files for output to a printer. *Pr* prints the named files on the standard output. If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing. If *file* is −, or if no files are specified, the standard input is assumed.

Columns are of equal width separated by at least one space; lines which do not fit are truncated, unless the **−s** option is used.

The width of an output line is 72 character positions for equal width multi-column output unless the **−w** option is used.

The length of an output page is 66 lines unless the **−l** option is used.

Pr advances to a new page using a sequence of line-feeds unless the **−f** option is used.

OPTIONS

The *options* may appear singly or be combined in any order.

- **+*k*** Begin printing with page *k*; the default is 1.
- **−*k*** Produce *k*-column output; the default is 1. The options **−e** and **−i** are assumed for multi-column output.
- **−a** Print multi-column output across the page. The **−*k*** must be specified for more than one column.
- **−d** Double-space the output.
- **−e*ck***
  Expand *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, *pr* assumes tab settings at every eighth position. *Pr* expands tab characters in the input into the appropriate number of spaces. The *c* (any non-digit character) designates the input tab character. If *c* is not specified, the tab character is used.
- **−f** Use the form-feed character for new pages; the default is to use a sequence of line feeds. Pause before beginning the first page if the standard output is associated with a terminal.
- **−h** Use the next argument as the header to be printed instead of the file name.
- **−i*ck*** Replace white space in output wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, *pr* assumes tab settings at every eighth position. The *c* (any non-digit character) designates the output tab character. If *c* is not specified, the tab character is used.
- **−l*k*** Set the length of a page to *k* lines; the default is 66.
- **−m** Merge and print all files simultaneously, one per column. This option overrides the **−*k*** and **−a** options.

−n*ck*
: Provide *k*-digit line numbering; the default for *k* is 5. The number occupies the first *k*+1 character positions of each column of normal output or each line of −**m** output. The *c* (any non-digit character) separates the line number from whatever follows. If *c* is not specified, a tab character is used.

−o*k* Offset each line by *k* character positions. The number of character positions per line is the sum of the width and offset.

−**p** Pause before beginning each page if the output is directed to a terminal. *Pr* rings the bell at the terminal and waits for a carriage return.

−**r** Print no diagnostic reports on failure to open files.

−s*c* Separate columns by the single character *c* instead of by the appropriate number of spaces. The default for *c* is a tab. Do not truncate lines.

−**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.

−w*k* Set the width of a line to *k* character positions; the default is 72 for equal-width multi-column output, no limit otherwise. The −**k** should be specified as something other than 1. If used with the −**m** option, the specified width must be greater than the number of files to be merged; for example, merging three files requires a minimum width of 4. If the −**w** option causes truncation, the −**s** option may not work.

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by **file list** and pipe the listing to a printer:

```
pr −3dh "file list" file1 file2 | print
```

Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr −e9 −t <file1 >file2
```

FILES

/dev/tty* to suspend messages

SEE ALSO

cat(1), pg(1).

SUPPORT STATUS

Supported.

NAME

print, lpr − line printer spooler

SYNOPSIS

print [options] files
lpr [options] files

DESCRIPTION

The *print* or *lpr* command spools the named file(s) for printing. The command creates an entry in the spool queue for each file listed on the input command line. The command also places a copy of the file to be printed in the spooler directory under a unique file name. After completion of the copy process, the command informs the shell to start the despooler which prints the file. The file created by the spooler for printing is deleted from the system upon completion of the print sequence.

The command does not to control pagination, headers, or any other part of the output. All control characters and formatting data must be placed in the file prior to invoking *print* or *lpr*.

If no files are named, the command reads from standard input.

OPTIONS

The options may appear in any order.

−b Do not print a banner page consisting of user name, date, filename and the contents of **/etc/lpmsg**. The default is to print the banner.

−cp nnn
Print *nnn* copies of this file. The range for *nnn* is 1 through 999. The default is 1.

−fm xxxxx
Use *xxxxx* as the forms name; the forms name is any 1-5 alphanumeric characters. The forms name specified here must agree with the forms name of the form specified as installed in the printer for printing to occur. See *spool*(1). The default is the standard print forms name **00000**.

−ln nn
Print *nn* number of lines per inch. Standard values are 6 and 8 lines per inch. The default is 6 lines per inch. This data is used by the printer operator to setup for special forms control and is not automatically used by the spooler.

−pr nn
Set *nn* as the priority for this print request. The range for nn is 0 through 15.

−pt lp*nn*
Redirect print output to the specified line printer. This option overides the default terminal/printer routing. The range for *nn* is 00 through the maximum number of printers allowed on the system.

EXAMPLES

To paginate and print report on the printer designated as **lp01** without a banner, enter

pr report | print −b −pt lp01

To print a previously formatted file of paychecks on a form designated as **paych**, enter

print −b −fm payck payfile

If the **payck** form is not designated as installed on the printer, the *payfile* is held in a wait state. After the **payck** form is designated as installed by using *spool*(1), *payfile* is printed.

MAPPING

Print provides mapping for up to ten device classes. Device class 0 maps tabs to appropriate spaces, etc. Device class 1 maps one to one. Device classes 2 through 9 may be defined by the user. The user must be the superuser, an analyst, or C programmer. See */usr/spool/lpd/oemdir/README* for information on defining device classes 2 through 9.

FILES

/usr/spool/lpd*	spool area
/usr/spool/lpd/lpd	despooler
/bin/print	spooler
/bin/lpr	spooler
/bin/spool	spool queue manager
/usr/spool/lpd/spooldev	spool device table manager
/usr/spool/lpd/??spldev	spool device table
/usr/spool/ldp/oemdir	user defined printer mapping
/usr/spool/lpd/??splque	spool queue
/usr/spool/lpd/sf*	spooled files
/etc/lpmsg	line printer message file

RESTRICTION

The *print* command queues a file and informs the shell to start the despooler. If no despooler is active and if the shell is terminated before the despooler is started, the files queued remain on the queue in a wait state. The files are printed during the next run of the despooler (i.e. the next print or spool −*start* command).

SEE ALSO

spooldev(1M), spool(1).

SUPPORT STATUS

Supported.

NAME

prof − display profile data

SYNOPSIS

prof [−**tcan**] [−**ox**] [−**g**] [−**z**] [−**h**] [−**s**] [−**m** mdata] [prog]

DESCRIPTION

Prof interprets a profile file produced by the *monitor*(3C) function. The symbol table in the object file *prog* (**a.out** by default) is read and correlated with a profile file (**mon.out** by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

A program creates a profile file if it has been loaded with the −**p** option of *cc*(1). This option to the *cc* command arranges for calls to *monitor*(3C) at the beginning and end of execution. It is the call to *monitor* at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the −**p** option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable PROFDIR. If PROFDIR does not exist, **mon.out** is produced in the directory current when the program terminates. If PROFDIR = string, "string/pid.progname" is produced, where *progname* consists of argv[0] with any path prefix removed, and *pid* is the program process id. If PROFDIR = nothing, no profiling output is produced.

A single function may be split into subfunctions for profiling by means of the MARK macro (see *prof*(5)).

OPTIONS

The mutually exclusive options **t**, **c**, **a**, and **n** determine the type of sorting of the output lines:

−**t** Sort by decreasing percentage of total time (default).

−**c** Sort by decreasing number of calls.

−**a** Sort by increasing symbol address.

−**n** Sort lexically by symbol name.

The mutually exclusive options **o** and **x** specify the printing of the address of each symbol monitored:

−**o** Print each symbol address in octal along with the symbol name.

−**x** Print each symbol address in hexadecimal along with the symbol name.

The following options may be used in any combination:

−**g** Include non-global symbols (static functions).

−**z** Include all symbols in the profile range (see *monitor*(3C)), even if associated with zero number of calls and zero time.

−h Suppress the heading normally printed on the report. This is useful if the report is to be processed further.

−s Print a summary of several of the monitoring parameters and statistics on the standard error output.

−m mdata
Use file *mdata* instead of **mon.out** as the input profile file.

FILES

mon.out for profile
a.out for namelist

SEE ALSO

cc(1), exit(2), profil(2), monitor(3C), prof(5).

WARNING

The times reported in successive identical runs may show variances of 20% or more because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may affect the data. In rare cases, the clock ticks initiating recording of the program counter may be in rhythm with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely, however.

RESTRICTIONS

Only programs that call *exit*(2) or return from *main* cause a profile file to be produced unless a final call to monitor is explicitly coded.

The use of the **−p** option of *cc*(1) to invoke profiling imposes a limit of 600 functions that may have call counters established during program execution. For more counters you must call *monitor*(3C) directly. If this limit is exceeded, other data is overwritten and the **mon.out** file is corrupted. The number of call counters used is reported automatically by the *prof* command whenever the number exceeds 5/6 of the maximum.

SUPPORT STATUS

Supported.

NAME

prs − print an SCCS file

SYNOPSIS

prs [−**d**[dataspec]] [−**r**[SID]] [−**e**] [−**l**] [−**c**[date-time]] [−**a**] files

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(4)) in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.), and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of options and file names.

OPTIONS

All the described options apply independently to each named file:

−**d**[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

−**r**[*SID*] Used to specify the *S*CCS *ID*entification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

−**e** Requests information for all deltas created *earlier* than and including the delta designated via the −**r** option or the date given by the −**c** option.

−**l** Requests information for all deltas created *later* than and including the delta designated via the −**r** option or the date given by the −**c** option.

−**c**[*date-time*] Cutoff date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

Units omitted from the date-time default to their maximum possible values; that is, −**c7502** is equivalent to −c750228235959. Any number of non-numeric characters may separate the various two-digit pieces of the *cutoff* date in the form: −**c77/2/2 9:22:25**.

−**a** Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1)) and existing, i.e., delta type = *D*, deltas. If the −**a** option is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(4)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords. A tab is specified by \t and carriage return/new-line is specified by \n. The default data keywords are:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	*yes* or *no*	S
:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

prs −d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

Users and/or user IDs for s.file are:
xyz
131
abc

prs −d"Newest delta for pgm :M:: :I: Created :D: By :P:" −r s.file

may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a *special case:*

prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta

for each delta table entry of the "D" type. The only option allowed to be used with the *special case* is the −a option.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(4).
Source Code Control System User Guide in the Support Tools Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

SUPPORT STATUS

Supported.

NAME

ps − report process status

SYNOPSIS

ps [options]

DESCRIPTION

Ps prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. The 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

Options using lists as arguments can have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

The *options* are:

−a Print information about all processes except process group leaders (see *intro*(2)) and processes not associated with a terminal.

−d Print information about all processes except process group leaders.

−e Print information about all processes.

−f Generate a *full* listing. See the OUTPUT DESCRIPTION below for the meaning of the columns in a full listing.

−g *grplist*
Restrict listing to data about processes whose process group leaders are given in *grplist*.

−l Generate a *long* listing. See OUTPUT DESCRIPTION.

−n *namelist*
Use the file *namelist* in place of the system namelist file **/unix**.

−p *proclist*
Restrict listing to data about processes whose process ID numbers are given in *proclist*.

−s *swapdev*
Use the file *swapdev* in place of **/dev/swap**. This is useful when examining a *corefile*; a *swapdev* of **/dev/null** causes the user block to be zeroed out.

−t *termlist*
Restrict listing to data about the processes associated with the terminals given in *termlist*. Terminal identifiers may be specified in one of two forms: the device file name (e.g., **tty04**) or if the device file name starts with **tty**, just the digit identifier (e.g., **04**).

−u *uidlist*
Restrict listing to data about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID is printed unless the **−f** option is used, in which case the login name is printed.

OUTPUT DESCRIPTION

The column headings and the meaning of the columns in a *ps* listing are given below; the letters f and l indicate the option full or long that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do *not* determine which processes are listed.

Heading	Option	Description
F	l	Process flags (octal and additive): 0 – swapped 1 – in core 2 – system process 4 – locked-in core 10 – being swapped 20 – being traced by another process 40 – another tracing flag
S	l	Process state: 0 – non-existent S – sleeping W – waiting R – running I – intermediate Z – terminated T – stopped X – growing
UID	f,l	User ID number of the process owner or login name under –f option
PID	all	Process ID of process; it is possible to kill a process using this ID
PPID	f,l	Process ID of parent process
C	f,l	Processor utilization for scheduling
PRI	l	Priority of process; higher numbers mean lower priority
NI	l	Nice value; used in priority computation
ADDR	l	Memory address of process if resident; otherwise, the disk address
SZ	l	Size in blocks of process core image
WCHAN	l	Event for which process is waiting or sleeping; if blank, process is running
STIME	f	Starting time of process
TTY	all	Controlling terminal for process
TIME	all	Cumulative execution time for process
CMD	all	Command name; full command name and arguments under –f option

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Under the –f option, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it

would appear without the −f option, is printed in square brackets.

FILES

/unix	system namelist
/dev/mem	memory
/dev/swap	the default swap device
/etc/passwd	supplies UID information
/etc/ps_data	internal data structure
/dev	searched to find terminal (tty) names

SEE ALSO

acctcom(1), kill(1), nice(1).

RESTRICTIONS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality.

Some data printed for defunct processes is irrelevant.

If the **/etc/ps_data** file is not current (i.e. after a kernel remake), *ps* gives invalid results. To make **/etc/ps_data** current, remove it and run *ps* again.

SUPPORT STATUS

Supported.

NAME

ptx − permuted index

SYNOPSIS

ptx [options] [input [output]]

DESCRIPTION

Ptx generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). *Ptx* has three phases:

1. Do the permutation generating one line for each keyword in an input line.
2. Rotate the keyword to the front and sort the permuted file.
3. Rotate the sorted lines so the keyword comes at the middle of each line.

Ptx output is in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx is assumed to be an *nroff*(1) or *troff*(1) macro provided by the user or provided by the *mptx*(5) macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as fits around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The index for this manual was generated using *ptx*.

OPTIONS

If the **−i** and **−o** options are missing, *ptx* uses /usr/lib/eign as the *ignore* file.

−f Fold upper and lower case letters for sorting.

−t Prepare the output for the phototypesetter.

−w *n* Use *n* as the length of the output line. The default line length is 72 characters for *nroff* and 100 for *troff*.

−g *n* Use *n* as the number of characters that *ptx* reserves in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.

−o *file* Use as keywords only the words given in *file*.

−i *file* Do not use as keywords any words given in *file*.

−b *file* Use the characters in *file* to separate words. Tab, newline, and space characters are *always* used as break characters.

−r Assume any leading non-blank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a fifth field on each output line.

FILES

/bin/sort

/usr/lib/eign
/usr/lib/tmac/tmac.ptx

SEE ALSO

nroff(1), troff(1), mm(5), mptx(5).

RESTRICTIONS

Line length counts do not account for overstriking or proportional spacing.

Lines that contain tildes (~) are not processed correctly because *ptx* uses that character internally.

SUPPORT STATUS

Supported.

NAME

pwd − working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

The messages

Cannot open ..
Read error in ..

indicate possible file system trouble and should be referred to the system administrator.

SUPPORT STATUS

Supported.

NAME

ratfor − rational Fortran dialect

SYNOPSIS

ratfor [options] [files]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:
{ statement; statement; statement }

decision-making:
if (condition) statement [**else** statement]
switch (integer value) {
case integer: statement
...
[**default**:] statement
}

loops:
while (condition) statement
for (expression; condition; expression) statement
do limits statement
repeat statement [**until** (condition)]
break
next

and some syntactic features to make programs easier to read and write:

free form input:
multiple statements/line; automatic continuation

comments:
this is a comment.

translation of relationals:
>, >=, etc., become .GT., .GE., etc.

return expression to caller from function:
return (expression)

define:
define *name replacement*

include:
include *file*

Ratfor is best used with *f77*(1).

OPTIONS

−h Turns quoted strings into **27H** constructs.

−C Copies comments to the output and attempts to format them neatly.

−6x Specifies *x* as the continuation character and places it in column 6. Ratfor normally marks continuation lines with an & in column 1.

SEE ALSO

f77(1).

Ratfor in the Programming Guide.

SUPPORT STATUS

Not supported.

NAME

regcmp − regular expression compile

SYNOPSIS

regcmp [−] files

DESCRIPTION

Regcmp, in most cases, precludes the need for calling *regcmp*(3X) from C programs. This reduces both execution time and program size.

The command *regcmp* compiles the regular expressions in *file* and places the output in *file*.i. If the − option is used, *regcmp* places the output in *file*.c. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *File*.i files may thus be *included* into C programs, or *file*.c files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex*(*abc*,*line*) applies the regular expression named *abc* to *line*.

EXAMPLES

```
name    "([A−Za−z][A−Za−z0−9_]*)$0"

telno   "\({0,1}([2−9][01][1−9])$0\){0,1} *"
        "([2−9][0−9]{2})$1[ −]{0,1}"
        "([0−9]{4})$2"
```

In the C program that uses the *regcmp* output,

```
regex(telno, line, area, exch, rest)
```

applies the regular expression named *telno* to *line*.

SEE ALSO

regcmp(3X).

SUPPORT STATUS

Supported.

NAME

rev − reverse lines of a file

SYNOPSIS

rev [file] ...

DESCRIPTION

Rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

SUPPORT STATUS

Supported.

NAME

rm, rmdir − remove files or directories

SYNOPSIS

rm [**−fri**] file ...

rmdir dir ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, the file permissions permissions are printed and a line is read from the standard input. If that line begins with y the file is deleted, otherwise the file remains. *Rm* or *rmdir* does not take this precaution if the **−f** option is given or if the standard input is not a terminal.

If a designated file is a directory, an error message is printed unless the option **−r** is used.

Rmdir removes entries for the named directories which must be empty.

OPTIONS

−f Remove files without verification from standard input (see DESCRIPTION above).

−r Recursively delete the entire contents of the specified directory and the directory itself.

−i Inquire whether to delete each file and, if the **−r** option is also used, whether to examine each directory.

EXAMPLES

To remove all files in the current directory which have file names beginning with **ch**, enter

```
rm ch*
```

Be careful using metacharacters (wildcards) so that you do not destroy files you mean to keep.

To remove the **oldstuff** directory and all files and subdirectories contained in the **oldstuff** directory, enter

```
rm −r oldstuff
```

To remove the **oldstuff** directory and all files and subdirectories contained in the **oldstuff** directory verifying each file removal, enter

```
rm −ir oldstuff
```

SEE ALSO

unlink(2).

NOTE

The file .. (dotdot) cannot be removed.

SUPPORT STATUS

Supported.

NAME

rmdel − remove a delta from an SCCS file

SYNOPSIS

rmdel −rSID files

DESCRIPTION

Rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file.

In addition, the delta specified must *not* be that of a version being edited for the purpose of making a delta; that is, if a *p-file* (see *get*(1)) exists for the named SCCS file, the delta specified must *not* appear in any entry of the *p-file*.

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

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User Guide*. Simply stated, they are

- if you make a delta you can remove it
- if you own the file and directory you can remove a delta

FILES

x.file	See *delta*(1)
z.file	See *delta*(1)

SEE ALSO

delta(1), get(1), help(1), prs(1), sccsfile(4).
Source Code Control System User Guide in the Support Tools Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

SUPPORT STATUS

Supported.

NAME

sact − print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the −e option has been previously executed without a subsequent execution of *delta*(1). If *file* is a directory, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of − is given, *sact* reads the standard input assuming each line is the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1	the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	the SID of the new delta to be created.
Field 3	the logname of the user who will make the delta (i.e. the user who executed a *get* for editing).
Field 4	the date that get −e was executed.
Field 5	the time that get −e was executed.

SEE ALSO

delta(1), get(1), unget(1).

DIAGNOSTICS

Use *help*(1) for explanations.

SUPPORT STATUS

Supported.

NAME

sag − system activity graph

SYNOPSIS

sag [options]

DESCRIPTION

Sag graphically displays the system activity data stored in a binary data file by a previous *sar*(1) run. Any of the *sar* data items may be plotted singly, or in combination; as cross plots, or versus time. Simple arithmetic combinations of data may be specified. *Sag* invokes *sar* and finds the desired data by string-matching the data column header (run *sar* to see what data is available).

OPTIONS

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. If *term* is **vpr**, output is processed by **vpr −p** and queued to a Versatec printer. Default for *term* is **$TERM**.

−x *spec* x axis specification with *spec* in the form given below.

Sag permits a single *spec* for the x axis. If unspeciifed, *time* is used.

−y *spec* y axis specification with *spec* in the same form as above. Up to 5 *spec*s separated by ; may be given for the y axis. The −y default is:

−y "%usr 00 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

AXIS SPECIFICATION

An axis specification is in the form

"name[op name]...[lo hi]"

Name is either an integer value, or a string that matches a column header in the *sar* report, with an optional device name in square brackets, e.g., **r+w/s[dsk−1]**.

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 *sag* evaluates A / A + B * 100 as (A/(A+B))*100, and A + B / C + D as (A+B)/(C+D).

Lo and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

Enclose the −x and −y arguments in double quotes if blanks or \<CR> are included in the specification.

EXAMPLES

To see CPU utilization for today:

```
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).

SUPPORT STATUS

Not supported.

NAME

sar − system activity reporter

SYNOPSIS

sar [−**ubdycwaqvmA**] [−**o** file] t [n]

sar [−**ubdycwaqvmA**] [−**s** time] [−**e** time] [−**i** sec] [−**f** file]

DESCRIPTION

Sar, using the first syntax, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. The default value of *n* is 1. If the −**o** option is specified, *sar* saves the samples in *file* in binary format.

Using the second syntax, with no sampling interval specified, **sar** extracts data from a previously recorded *file*, either the one specified by the −**f** option or, if −**f** is not used, 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.

OPTIONS

The following options, valid in both usages of *sar*, specify the subsets of data to be printed. Listed under each option are column headings, which appear when the option is invoked, and their meanings. If no options are specified, the −**u** option is assumed.

−**u** Report CPU utilization.
- *%usr* − Portion of time running in user mode.
- *%sys* − Portion of time running in system mode.
- *%wio* − Portion of time idle with some process waiting for block I/O.
- *%idle* − Portion of time otherwise idle.

−**b** Report buffer activity:
- *bread/s*, *bwrit/s* − Transfers per second of data between system buffers and disk or other block devices.
- *lread/s*, *lwrit/s* − Accesses of system buffers.
- *%rcache*, *%wcache* − Cache match ratios, e. g., 1 − bread/lread.
- *pread/s*, *pwrit/s* − Transfers via raw device mechanism.

−**d** Report activity for each block device, e. g., disk or tape drive:
- *%busy* − Portion of time device was busy servicing a transfer request.
- *avque* − Average number of requests outstanding during *%busy*.
- *r+w/s* − Number of data transfers to or from a device.
- *blks/s* − Number of bytes transferred to or from a device in 512 byte units.
- *avwait* − Average time in ms. that transfer requests wait idly on queue.
- *avserv* − Average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

−**y** Report TTY device activity:

- *rawch/s* — Input character rate.
- *canch/s* — Input character rate processed by canon.
- *outch/s* — Output character rate.
- *rcvin/s* — Receive interrupt rates.
- *xmtin/s* — Transmit interrupt rates.
- *mdmin/s* — Modem interrupt rates.

−c Report system calls:
- *scall/s* — System calls of all types.
- *sread/s*, *swrit/s*, *fork/s*, *exec/s* — Specific system calls.
- *rchar/s*, *wchar/s* — Characters transferred by read and write system calls.

−w Report system swapping and switching activity:
- *swpin/s*, *swpot/s* — Number of transfers for swapins (including initial loading of some programs) and swapouts.
- *bswin/s*, *bswot/s* — Number of 512 byte units transferred for swapins (including initial loading of some programs) and swapouts.
- *pswch/s* — Process switches.

−a Report use of file access system routines: how many times per second the *iget/s*, *namei/s*, *dirblk/s* routines were called.

−q Report average queue length while occupied, and percent (%) of time occupied:
- *runq-sz*, *%runocc* — Run queue of executable processes in memory.
- *swpq-sz*, *%swpocc* — Swap queue of processes swapped out but ready to run.

−v Report status of text, process, inode and file tables:
- *text-sz*, *proc-sz*, *inod-sz*, *file-sz* — Entries/size for each table, evaluated once at sampling point.
- *text-ov*, *proc-ov*, *inod-ov*, *file-ov* — Overflows occurring between sampling points.

−m Report message and semaphore activities:
- *msg/s*, *sema/s* — Primitives per second.

−A Report all data. Equivalent to **−udqbwcayvm.**

EXAMPLES

To see CPU activity so far for today, enter

```
sar
```

To watch CPU activity evolve for 10 minutes and save data, enter

```
sar −o temp 60 10
```

FILES

/usr/adm/sa/sa*dd* daily data file

RESTRICTIONS

The **−d** option does not work.

SEE ALSO

sag(1G), sar(1M).

SUPPORT STATUS

Not supported.

NAME

sccsdiff − compare two versions of an SCCS file

SYNOPSIS

sccsdiff −rSID1 −rSID2 [−p] [−sn] files

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but options apply to all files.

OPTIONS

−r*SID1* −r*SID2*
: Specifies *SID1* and *SID2* as the deltas of an SCCS file that are to be compared. *Sccsdiff* passes versions to *bdiff*(1) in the order given.

−p
: Pipes output for each file through *pr*(1).

−s*n*
: Denotes *n* as the file segment size that *bdiff* passes to *diff*(1). This is useful when *diff* fails due to a high system load.

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1).
Source Code Control System in the *Support Tools Guide*.

DIAGNOSTICS

The message

file: No differences

is printed if the two versions are the same.

Use *help*(1) for explanations.

SUPPORT STATUS

Supported.

NAME

sdb − symbolic debugger

SYNOPSIS

sdb [−w] [−W] [objfil [corfil [directory-list]]]

DESCRIPTION

Sdb is a symbolic debugger that can be used with C and F77 programs. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

Objfil is normally an executable program file which has been compiled with the −g (debug) option; if it has not been compiled with the −g option, or if it is not an executable file, the symbolic capabilities of *sdb* are limited, but the file can still be examined and the program debugged. The default for *objfil* is **a.out**. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is **core**. The core file need not be present. A − in place of *corfil* forces *sdb* to ignore any core image file. The colon separated list of directories (*directory-list*) is used to locate the source files used to build *objfil*.

It is useful to know that at any time there is a *current line* and *current file*. If *corfil* exists then they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in *main*(). The current line and file may be changed with the source file examination commands.

By default, warnings are provided if the source files used in producing *objfil* cannot be found or are newer than *objfil*. This checking feature and the accompanying warnings may be disabled by the use of the −W option.

Names of variables are written just as they are in C or F77. Note that names in C are of arbitrary length; *sdb* does not truncate names. Variables local to a procedure may be accessed using the form *procedure:variable*. If no procedure name is given, the procedure containing the current line is used by default.

It is also possible to refer to structure members as *variable.member*, pointers to structure members as *variable−>member*, and array elements as *variable*[*number*]. Pointers may be dereferenced by using the form *pointer*[0]. Combinations of these forms may also be used. F77 common variables may be referenced by using the name of the common block instead of the structure name. Blank common variables may be named by the form *.variable*. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by *sdb*. An unqualified structure variable may also be used with various commands. Generally, *sdb* interprets a structure as a set of variables. Thus, *sdb* displays the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value *sdb* displays, not the addresses of

individual elements.

Elements of a multidimensional array may be referenced as *variable*[*number*]/**number** or as *variable*[*number,number,...*]. In place of *number*, the form *number;number* may be used to indicate a range of values, * may be used to indicate all valid values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, *sdb* displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts are omitted. A multidimensional parameter in an F77 program cannot be displayed as an array, but it is actually a pointer, whose value is the location of the array. The array itself can be accessed symbolically from the calling function.

A particular instance of a variable on the stack may be referenced by using the form *procedure:variable,number*. All the variations mentioned in naming variables may be used. *Number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal or hexadecimal.

Line numbers in the source program are referred to as *filename:number* or *procedure:number*. In either case the number is relative to the beginning of the file. If no procedure or file name is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under *sdb*, all addresses refer to the executing program; otherwise they refer to *objfil* or *corfil*. An initial argument of −w permits overwriting locations in *objfil*.

Individual processor general registers may be named instead of variables by using the register name with a % prepended. The x command displays the current values of all the general registers. The contents of these registers can be displayed or modified.

Note that the hardware floating point registers of the 68881 math coprocessor are also available to be used as the 68020 general registers when the 68881 math coprocessor installed. These registers are named **%fp0** through **%fp7**.

OPTIONS

−w Permit overwriting locations in *objfil*.

-W Disable warnings.

ADDRESSES

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

b1 ≤*address*<*e1* => *file address*=*address*+*f1*−*b1*
otherwise

b2 ≤*address*<*e2* => *file address*=*address*+*f2*−*b2*

otherwise, the requested *address* is not valid. In some cases (e.g., for programs with separated I and D space) the two segments for a file may overlap.

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

In order for *sdb* to be used on large files, all appropriate values are kept as signed 32-bit integers. The M command can be used to display or change the current values for the address maps.

COMMANDS

The commands for examining data in the program are:

t Print a stack trace of the terminated or halted program.

T Print the top line of the stack trace.

variable|*clm*

Print the value of *variable* according to length *l* and format *m*. A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

- **b** one byte
- **h** two bytes (half word)
- **l** four bytes (long word)

Valid values for *m* are:

- **c** Character
- **d** Decimal
- **u** Decimal, unsigned
- **o** Octal
- **x** Hexadecimal
- **f** 32-bit single precision floating point
- **g** 64-bit double precision floating point
- **s** Assume *variable* is a string pointer and print characters starting at the address pointed to by the variable.
- **a** Print characters starting at the variable address. This format may not be used with register variables.
- **p** Pointer to procedure
- **i** Disassemble machine-language instruction with addresses printed numerically and symbolically.
- **I** Disassemble machine-language instructions with addresses just printed numerically.

The length specifiers are only effective with the formats **c**, **d**, **u**, **o** and **x**. Any of the specifiers, *c*, *l*, and *m*, may be omitted. If all are omitted, *sdb* choses a length and a format suitable for the variable type as declared in the program. If *m* is

specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells *sdb* to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or if no length is given, by the size associated with the *variable*. If a count specifier is used for the **s** or **a** command, then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command *./*.

The *sh*(1) metacharacters * and ? may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, variables local to the current procedure and global variables are matched; if a procedure name is specified then only variables local to that procedure are matched. To match only global variables, the form *:pattern* is used.

linenumber?*lm*
variable:?*lm*
Print the value at the address from a.out or I space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is "i".

variable=*lm*
linenumber=*lm*
number=*lm*
Print the address of *variable* or *linenumber*, or the value of *number*, in the format specified by *lm*. If no format is given, then **lx** is used. The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

variable!*value*
Set *variable* to the given *value*. The value may be a number, a character constant or a variable. The value must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted '*character*. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The *variable* may be an expression which indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C conventions are used in any type conversions necessary to perform the indicated assignment.

x Print the machine registers and the current machine-language instruction.

X Print the current machine-language instruction.

The commands for examining source files are:

e *procedure*
e *file-name*
e *directory/*
e *directory file-name*

The first two forms set the current file to the file containing *procedure* or to *file-name*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, file name, or directory is given, the current procedure name and file name are reported.

/regular expression/

Search forward from the current line for a line containing a string matching *regular expression* as in *ed*(1). The trailing / may be deleted.

?regular expression?

Search backward from the current line for a line containing a string matching *regular expression* as in *ed*(1). The trailing ? may be deleted.

p Print the current line.

z Print the current line followed by the next 9 lines. Set the current line to the last line printed.

w Window. Print the 10 lines around the current line.

number

Set the current line to the given line number. Print the new current line.

count+

Advance the current line by *count* lines. Print the new current line.

count−

Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

count r *args*
count R

Run the program with the given arguments. The r command with no arguments reuses the previous arguments to the program while the R command runs the program with no arguments. An argument beginning with < or > causes redirection for the standard input or output, respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

linenumber c *count*
linenumber C *count*

Continue after a breakpoint or interrupt. If *count* is given, it specifies the breakpoint at which to stop after ignoring *count* − 1 breakpoints. C continues with the signal which caused

the program to stop reactivated and **c** ignores it. If a line number is specified then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes.

linenumber **g** *count*
Continue after a breakpoint with execution resumed at the given line. If *count* is given, it specifies the number of breakpoints to be ignored.

s *count*
S *count*
Single step the program through *count* lines. If no count is given then the program is run for one line. **S** is equivalent to **s** except it steps through procedure calls.

i
I Single step by one machine-language instruction. **I** steps with the signal which caused the program to stop reactivated and **i** ignores it.

variable$**m** *count*
address:**m** *count*
Single step (as with **s**) until the specified location is modified with a new value. If *count* is omitted, it is effectively infinity. *Variable* must be accessible from the current procedure. Because this command is done by software, it can be very slow.

level **v**
Toggle verbose mode, for use when single stepping with **S**, **s** or **m**. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each C source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A **v** turns verbose mode off if it is on for any level.

k Kill the program being debugged.

procedure(arg1,arg2,...)
procedure(arg1,arg2,...)/*m*
Execute the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to **d**.

linenumber **b** *commands*
Set a breakpoint at the given line. If a procedure name without a line number is given (e.g., "proc:"), a breakpoint is placed at the first line in the procedure even if it was not compiled with the **−g** option. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given, execution stops just before the breakpoint and control is returned to *sdb*. Otherwise the *commands* are executed

when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If **k** is used as a command to execute at a breakpoint, control returns to *sdb*, instead of continuing execution.

B Print a list of the currently active breakpoints.

linenumber **d**
Delete a breakpoint at the given line. If no *linenumber* is given then the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a **y** or **d** then the breakpoint is deleted.

D Delete all breakpoints.

l Print the last executed line.

linenumber **a**
Announce. If *linenumber* is of the form *proc:number*, the command effectively does a *linenumber* **b l**. If *linenumber* is of the form *proc:*, the command effectively does a *proc:* **b T**.

Miscellaneous commands:

!*command*
The command is interpreted by *sh*(1).

new-line
If the previous command printed a source line then advance the current line by one line and print the new current line. If the previous command displayed a memory location, then display the next memory location.

control-d
Scroll. Print the next 10 lines of instructions, source or data depending on which was printed last.

< *filename*
Read commands from *filename* until the end of file is reached, and then continue to accept commands from standard input. When *sdb* is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; < may not appear as a command in a file.

M Print the address maps.

M [?/][*] *b e f*
Record new values for the address map. The arguments ? and / specify the text and data maps, respectively. The first segment, (*b1, e1, f1*), is changed unless * is specified, in which case the second segment (*b1, e1, f1*), of the mapping is changed. If fewer than three values are given, the remaining map parameters are left unchanged.

" *string*
Print the given string. The C escape sequences of the form *character* are recognized, where *character* is a nonnumeric character.

q Exit the debugger.

The following commands also exist and are intended only for debugging the debugger:

V Print the version number.
Q Print a list of procedures and files being debugged.
Y Toggle debug output.

FILES

a.out
core

SEE ALSO

cc(1), f77(1), sh(1), a.out(4), core(4).
Symbolic Debugging Program — "*sdb*" in the Programming Guide.

WARNINGS

When *sdb* prints the value of an external variable for which there is no debugging information, a warning is printed before the value. The value is assumed to be int (integer).

Data which are stored in text sections are indistinguishable from functions.

Line number information in optimized functions is unreliable, and some information may be missing. Note that *cc*(1) disables optimization for modules compiled with the −g option.

RESTRICTIONS

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

The default type for printing F77 parameters is incorrect. Their address is printed instead of their value.

Tracebacks containing F77 subprograms with multiple entry points may print too many arguments in the wrong order, but their values are correct.

The range of an F77 array subscript is assumed to be *1* to *n*, where *n* is the dimension corresponding to that subscript. This is only significant when the user omits a subscript or uses * to indicate the full range. There is no problem in general with arrays having subscripts whose lower bounds are not 1.

SUPPORT STATUS

Supported.

NAME

sdiff − side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

Sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x       |       y
a               a
b       <
c       <
d               d
        >       c
```

OPTIONS

−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*.

Sdiff copies identical lines of *file1* and *file2* to *output*. *Sdiff* prints sets of differences, as produced by *diff*(1); 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).

SUPPORT STATUS

Supported.

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.

A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

Address, *function*, and *argument* are described below.

In normal operation, *sed* cyclically copies a line of input into an internal *edit buffer* and applies in sequence all commands whose *addresses* apply to the lines within the edit buffer. Editing commands can be applied only to non-selected pattern spaces by use of the negation function ! (below). At the end of the script, *sed* copies the pattern space to the standard output and deletes the pattern space. Some of the commands use an internal *temporary buffer* to save all or part of the *edit buffer* for subsequent retrieval.

OPTIONS

The following options can be specified several times.

−f *sfile* Takes the script from *sfile*.

−e *script* Specifies *script* as the script. If there is just one **−e** option and no **−f** option, the **−e** may be omitted. Script should be surrounded by quotes to isolate it from the shell.

−n Suppresses the default output.

ADDRESS

An *address* is either a decimal number that counts input lines cumulatively across files, or a $ that addresses the last line of input. An address may also be a context address, i.e., a */regular expression/* in the style of *ed*(1), with the following differences:

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.

FUNCTION

In the following list of functions the maximum number of permissible addresses for each function is indicated in the parentheses preceding each function.

(1) **a**
text Append. Place *text* to the output before reading the next input line (see *text* below).

(2) **b** *label* Branch to the : command bearing the *label*. If *label* is not specified, branch to the end of the script.

(2) **c**
text Change. Delete the edit buffer. With 0 or 1 address or at the end of a 2-address range, place *text* on the output (see *text* below). Start the next cycle.

(2) **d** Delete the edit buffer. Start the next cycle.

(2) **D** Delete the initial segment of the edit buffer through the first new-line. Start the next cycle; do not copy a new line into the pattern space if any lines remain in the pattern space.

(2) **g** Replace the contents of the edit buffer by the contents of the temporary buffer.

(2) **G** Append the contents of the temporary buffer to the edit buffer.

(2) **h** Replace the contents of the temporary buffer by the contents of the edit buffer.

(2) **H** Append the contents of the edit buffer to the temporary buffer.

(1) **i**
text Insert. Place *text* on the standard output. (see *text* below)

(2) **l** List the edit buffer on the standard output in an unambiguous form. Spell non-printing characters in two-digit ASCII and fold long lines.

(2) **n** Copy the edit buffer to the standard output. Replace the edit buffer with the next line of input.

(2) **N** Append the next line of input to the edit buffer with an embedded new-line. (The current line number changes.)

(2) **p** Print. Copy the edit buffer to the standard output.

(2) **P** Copy the initial segment of the edit buffer through the first new-line to the standard output.

(1) **q** Quit. Branch to the end of the script. Do not start a new cycle.

(2) r *rfile* Read the contents of *rfile*. Place the contents of *rfile* on the output before reading the next input line. (see *rfile* below)

(2) s/*regular expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular expression* in the edit buffer. 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 edit buffer if a replacement was made.
- **w** *wfile* Write. Append the edit buffer to *wfile* if a replacement was made. (see *wfile* below)

(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 edit buffer to *wfile*. (see *wfile* below)

(2) x Exchange the contents of the pattern and temporary buffers.

(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*
Apply the *function* (or group, if *function* is {) only to lines *not* selected by the address(es).

(0): *label* Specify a *label* to which b and t commands may branch.

(1) = Place the current line number on the standard output as a line.

(2) { Execute the following commands through a matching } only when the edit buffer is selected.

(0) An empty command is ignored.

(0) # If a # is the first character on the first line of a script file, the entire line is treated as a comment unless the character following the # is an n. If an n follows the #, the default output is suppressed and the rest of the line after the #n is ignored. A script file must contain at least one non-comment line.

ARGUMENT

text One or more lines of text; each line must end with a backslash to escape the new-line, except the last line. Backslashes within text are escape characters, and may be used to retain initial tabs and blanks that *sed* usually strips from every script line.

rfile Read file; this argument must be the last one in the command line and exactly one blank must separate it from its *function*.

wfile Write file; this argument must be the last one in the command line and exactly one blank must separate it from its *function*. *Sed* creates each *wfile* (up to ten distinct *wfiles*) before processing.

SEE ALSO

awk(1), ed(1), grep(1).

SUPPORT STATUS

Supported.

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 | (or, for historical compatibility, by ^). The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. 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. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

The separators affect execution as follows:

; Sequentially execute the preceding pipeline.

& Asynchronously execute the preceding pipeline; that is, the shell does not wait for the pipeline to finish execution before proceeding to execute the next command.

&& Execute the following *list* only if the preceding pipeline returns a zero exit status.

|| Execute the following *list* only if the preceding pipeline returns a non-zero exit status.

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*;}
Simply execute *list*.

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 standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing new-lines

are removed.

Parameter Substitution

The character $ is used to introduce substitutable *parameters*. 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:

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.
- - Options 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*(1) 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*(1) 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 checks 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 checks 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 newline.

SHACCT
If this parameter is set to the name of a file writable by the user, the shell writes an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(1) and *acctcms*(1M) can be used to analyze the data collected.

SHELL
When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and there is an **r** in 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.

File Name Generation
Following substitution, 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 \. The pair \new-line is ignored. All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, $. "$*" is equivalent to "$1 $2 ...", whereas "$@" is equivalent to "$1" "$2" ....

Prompting

When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt, the value of PS2, is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

- **<word** Use file *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** Use the shell input (read up to a line that is the same as *word*, or to an end-of-file) as the standard input. If any character of *word* is quoted, no interpretation is placed upon the input characters; otherwise, parameter and command substitution occurs, (unescaped) \new-line is ignored, and \ must be used to quote the characters \, $, *word*. If − is appended to <<, all leading tabs are stripped from

word and from shell input.

<&digit Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using >&digit.

<&− Close the standard input. Similarly for the standard output using >&−.

If any of the above is preceded by a digit, the file descriptor which is 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*.

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.

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 parameter to the environment (see also *Invocation* **−a**). A parameter may be removed from the environment with the **unset**command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

TERM=450 cmd and
(export TERM; TERM=450; cmd)

are equivalent as far as the execution of *cmd* is concerned.

If the **−k** option is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The

following first prints a=b c and c:

```
echo a=b c
set -k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent, 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 or between the colon delimiters anywhere else in the path list. If the command name contains a / the search path is not used; such commands are not 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 *exec*s 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 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* ...]
Read the arguments as input to the shell and execute the resulting command(s).

exec [*arg* ...]
Execute the command specified by the arguments 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*]
Exit a shell with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed (an end-of-file also causes the shell to exit.)

export [*name* ...]
Mark the given *name*s for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may *not* be exported.

hash [**−r**] [*name* ...]
For each *name*, determine and remember the location in the search path of the command specified by *name*. 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. There are certain situations which require that the stored location of a command be recalculated. Commands for which this is done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* is incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to exec **newgrp** *arg* .... See *newgrp*(1) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]
Read one line from the standard input and assign the first word to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
Mark the given *names readonly;* 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*]
Exit a function with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [−−**aefhkntuvx** [*arg* ...]]
Set options and positional parameters. See *Invocation* for a description of the options. Using + rather than − causes these options to be turned off. The current set of options 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*] ...
Read the command *arg* and execute when signal(s) *n* is received. 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 child processes (files of any size may be read). With no argument, the current limit is printed.

umask [*nnn*]
Set the user file-creation mask to *nnn* (see *umask*(2)). 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 the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

Invocation

If the shell is invoked through *exec*(2) and the first character of argument zero is −, commands are initially read from /etc/profile and 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 options below are interpreted by the shell on invocation. Note that unless the −c or −s option is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file.

−a Mark variables which are modified or created for export.

−c *string*
Read commands from *string*.

−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.

−i If the −i option is present or if the shell input and output are attached to a terminal, make this shell *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.

−k Place all keyword arguments in the environment for a command, not just those that precede the command name.

−n Read commands but do not execute them.

−r Make the shell a restricted shell.

−s If the −s option is present or if no arguments remain, read commands from the standard input. Any remaining arguments specify the positional parameters. Shell output except for *Special Commands* is written to file descriptor 2.

−t Exit after reading and executing one command.

−u Treat unset variables as an error when substituting.

−v Print shell input lines as they are read.

−x Print commands and their arguments as they are executed.

−− Do not change any of the options; useful in setting $1 to −.

Rsh Only

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of $PATH,
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The superuser often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some superusers also supply *rsh* users with the 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

acctcom(1), cd(1), echo(1), env(1), login(1), newgrp(1), pwd(1), test(1), umask(1), acctcms(1M), dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), umask(2), wait(2), a.out(4), profile(4), environ(5).
Shell Tutorial in the User Guide.
Shell Introduction, *Using Shell Commands*, *Shell Programming*, and *Examples of Shell Procedures* in the Programming Guide.

WARNINGS

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 continues to *exec*(2) 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.

SUPPORT STATUS

Supported.

NAME

shl − shell layer manager

SYNOPSIS

shl

DESCRIPTION

Shl 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. The output of a layer may be blocked when the layer is not current by setting the **loblk** option of *stty*(1) or by issuing the **block** command of *shl*.

The *stty* control-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.

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. These names 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 is 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 is 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), ioctl(2), signal(2), sxt(7).

RESTRICTIONS
Shl can not be used as a login shell.

Shl can only be invoked when the system is in the multi-user mode. If *shl* is invoked while the system is in the single-user mode, the message 'Multiplex failed (errno = 25)' is displayed.

WARNINGS
Pressing the Rubout or equivalent key while at the *shl* level prompt, may cause the entire line, including the shell prompt >>>, to be erased and positions the cursor on the next line. The shell prompt is not displayed but a command can be entered on the blank line. Pressing the Return or equivalent key causes the shell prompt >>> to be displayed.

SUPPORT STATUS
Supported.

NAME

size − print section sizes of common object files

SYNOPSIS

size [−**o**] [−**x**] [−**V**] files

DESCRIPTION

The *size* command produces section size information for each section in the common object files. *Size* prints the size of the text, data and bss (uninitialized data) sections along with the total size of the object file. If an archive file is input to the *size* command, the information for all archive members is displayed.

Size prints numbers in decimal unless either the −**o** or the −**x** option is used.

OPTIONS

−**o** Print numbers in octal

−**x** Print numbers in hexadecimal

−**V** Print the version information

NOTE

Release 2.x archive format is supported permitting transparent use with archive libraries from Release 2.x.

SEE ALSO

as(1), cc(1), ld(1), a.out(4), ar(4).

DIAGNOSTICS

size: name: cannot open
name cannot be read

size: name: bad magic
name is not an appropriate common object file

SUPPORT STATUS

Supported.

NAME

sleep − suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep suspends execution for *time* seconds. *Sleep* executes a command after a certain amount of time:

```
(sleep 105; command)&
```

or executes a command every so often:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

alarm(2), sleep(3C).

RESTRICTIONS

Time must be less than 65536 seconds. It is also recommended that *time* be greater than 1.

SUPPORT STATUS

Supported.

NAME

sln – link files symbolically

SYNOPSIS

sln file1 [file2 ...] target

DESCRIPTION

File1 is linked symbolically 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 linked to that directory.

If the *sln* command is invoked by the super-user then *file1* may be a directory.

Sln is similar to the *ln(1)* command, except for the following:

–

File1 and *target* can exist on different file systems.

–

If the *target* is removed after the *sln*(1) command has been issued, then any file access attempts will result in an error.

The *rm*(1) command must be used on all the files that were remotely linked to *target.*

SEE ALSO

ln(1), rm(1), unlink(1M), slink(2).

RESTRICTIONS

If *file1* is a directory or has subsequently been removed then the *unlink*(1M) command must be invoked to remove the symbolic link.

SUPPORT STATUS

Supported.

NAME

sno – SNOBOL interpreter

SYNOPSIS

sno [files]

DESCRIPTION

Sno is a compiler and interpreter very similar to SNOBOL. The slight differences are listed under **DIFFERENCES**.

Input to *sno* is the concatenation of the named *files* and the standard input. *Sno* considers input statements, up to and including the label **end**, a program, and compiles those statements. All other statements are available to **syspit**.

DIFFERENCES

Sno has no unanchored searches. To get the same effect:

```
a ** b          unanchored search for b.
a *x* b = x c   unanchored assignment
```

Sno has no back referencing.

```
x = "abc"
a *x* x         is an unanchored search for abc.
```

Sno declares functions at compile time using the (non-unique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and *sno* preempts the use of the name **define**. *Sno* does not provide for automatic variables other than parameters. Examples:

```
define f( )
define f(a, b, c)
```

All labels except **define** (even **end**) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on **end** cannot merely name a label.

If **start** is a label in the program, *sno* starts program execution there. If not, *sno* begins execution with the first executable statement; **define** is not an executable statement.

Sno has no builtin functions.

Sno does not need parentheses for arithmetic; normal precedence applies. Because of this, spaces must set off the arithmetic operators / and *.

The right side of assignments must be non-empty.

Either ' or " may be used for literal quotes.

The pseudo-variable **sysppt** is not available.

SEE ALSO

awk(1).

SUPPORT STATUS

Not supported.

NAME

sort − sort and/or merge files

SYNOPSIS

sort [−**cmu**] [−**o**output] [−**y**kmem] [−**z**recsz] [−**dfiMnr**] [−**btx**] [+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*

Use the argument given as the name of an output file 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* starts using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum is 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** option, a popular system default size is used. Lines longer than the buffer size cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) prevents 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 characters outside the ASCII range 040-0176 in non-numeric comparisons.

−M Compare as months. The first three non-blank characters of the field are folded to upper case and compared so that JAN < FEB < ... < DEC. Invalid fields compare low to JAN. The **−M** option implies the **−b** option (see below).

−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 below). 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 below), 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 at *pos2*. The characters at positions *pos1* and *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.

−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 (e.g., *xx* delimits an empty field).

−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 is applied to all *+pos1* arguments. Otherwise, the **b** option may be attached independently to each *+pos1* or *−pos2* argument (see below).

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the options **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** option 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** option 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).

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (e.g., 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.

RESTRICTIONS

Sort outputs files without truncation if the file has up to 46,380 lines of up to 1024 characters per line. *Sort* outputs files with truncation (and displays an error message) if the file exceeds 12,488 characters or 1561 lines and the line length exceed 1024 characters.

SUPPORT STATUS

Supported.

NAME

spell, hashmake, spellin, hashcheck − find spelling errors

SYNOPSIS

spell [−v] [−b] [−x] [−l] [−i] [+local_file] [files]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

Spell collects words from the named *files* and looks them up in a spelling list. *Spell* prints words on the standard output that do not occur in the spelling list, and cannot be derived from words in that list by applying certain inflections, prefixes, and/or suffixes. If no *files* are named, *spell* collects words from the standard input.

Spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Spell follows chains of included files (**.so** and **.nx** *troff*(1) requests), *unless* the names of such included files begin with **/usr/lib**.

The spelling list is based on many sources, and while it is less complete than an ordinary dictionary in some ways, it 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*). *Spell* accumulates copies of all output 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 n Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output. Information about the hash coding is printed on standard error.

hashcheck Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; *hashcheck* writes these codes on the standard output.

OPTIONS

−v Print all words not literally in the spelling list, with plausible derivations from the words in the spelling list indicated.

−b Check British spelling, preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc. This option also assumes that all ize suffixes should be spelled ise; this assumption is incorrect.

−x Print every plausible stem with = for each word.

−l Follow the chains of all files including those that begin with **/user/lib.**

−i Ignore all chains of included files.

+local_file
Remove words found in *local_file* from the output of spell. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. This option permits the user to specify a set of words that are correct spellings, in addition to the spelling list used by *spell*.

EXAMPLE

The following example creates the hashed spell list *hlist* and checks the result by comparing the two temporary files; they should be equal.

```
cat goodwds | /usr/lib/spell/hashmake | sort −u >tmp1
cat tmp1 | /usr/lib/spell/spellin `cat tmp1 | wc −l` >hlist
cat hlist | /usr/lib/spell/hashcheck >tmp2
diff tmp1 tmp2
```

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American and 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

eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(1).

RESTRICTIONS

The coverage of the spelling list is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these additions are kept in a separate local file that is added to the hashed *spelling_list* by *spellin*.

SUPPORT STATUS

Supported.

NAME

spline − interpolate smooth curve

SYNOPSIS

spline [options]

DESCRIPTION

Spline takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed., pp. 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph*(1G).

OPTIONS

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

-k *k* Set *k* to be constant k (default *k* is 0) used in the boundary value computation:

$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$

-n *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 *low* [*high*]
Denotes *low* as the lower *x* limit; denotes *high* (if given) as the upper *x* limit. 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.

RESTRICTIONS

A limit of 1,000 input points is enforced silently.

SUPPORT STATUS

Supported.

NAME

split − split a file into pieces

SYNOPSIS

split [**−n**] [**−b**] [file [name]]

DESCRIPTION

Split reads *file* and writes it into 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 the default.

If no input file is given, or if − is given in its place, the standard input file is used.

OPTIONS

−n Number of pieces in each output file; the default is 1000.

−b Pieces are 512-byte blocks; the default is that pieces are lines.

EXAMPLE

To split the **toobig** file into 500 line files named **smallaa**, **smallab**, **smallac**, etc., enter

```
split −n 500 toobig small
```

SEE ALSO

bfs(1), csplit(1).

SUPPORT STATUS

Supported.

NAME

spool − spool queue manager

SYNOPSIS

spool command-option secondary-option ...

DESCRIPTION

Spool controls the spool queue after a file has been placed into the spool queue by *print*(1) or *lpr* for printing. *Spool*:

- Controls printing through the ability to purge, hold, restart, start, and stop the printer spooler system.
- Allows the user to query and modify the file entries in the spool queue.
- Allows update of certain control data by accessing the spool queue.

Only the originator of the file or the superuser may change data in the spool queue or view the spooled file.

COMMAND SYNTAX

The following table lists the *spool* command options and secondary options.

Command	Command Options	Secondary Options
spool	−help	
spool	−query	
spool	−change	−sp spool_id [−pt lpnn] [−pr xx] [−fm xxxxx]
spool	−purge	−sp spool_id [−ws tnn] [−pt lpnn] [−who xxxxxxxx]
spool	−hold	−sp spool_id [−ws tnn] [−pt lpnn] [−who xxxxxxxx]
spool	−release	−sp spool_id [−ws tnn] [−pt lpnn] [−who xxxxxxxx]
spool	−start	−pt lpnn spool
spool	−look	−sp spool_id
spool	−restart	−pt lpnn [−pg nnn]

SECONDARY OPTIONS

Secondary options have the following meanings and values:

Option	Description
−sp spool-id	Spool identification: spool-id is sf followed by six digits
−cp xxx	Number of copies: xxx is 1-999
−fm xxxxx	Form name: xxxxx is 1-5 alphanumeric characters
−pg nnnn	Page number: nnnn is 1-9999
−pr xx	Priority: xx is 0-15
−pt lpnn	Printer number: nn is 00-maximum number of printers allowed
−who xxxxxxxx	User name: xxxxxxxx is 1-8 characters
−ws tnn	Terminal number: nn is 00-15

COMMAND OPTIONS

−help

The *help* option displays the list of commands on the screen. No secondary options are used.

−query

The *query* option displays the current spool queue status. This display is screen oriented and is available to all users. No secondary options are used.

−change

The *change* option allows the user to change specified spooled file values. The entries in the spool queue can be modified by the superuser or the originating user. The various forms of this command are listed below:

spool −change −sp spool_id −pt lp02

Change the destination printer to lp02 for the file specified by spool_id.

spool −change −sp spool_id −pr nn

Change the print priority to nn for the file specified by spool_id.

spool −change −sp spool_id −cp nn

Change the number of copies to nn for the file specified by spool_id.

spool −change −sp spool_id −fm xxxxx

Change the forms name to xxxxx for the file specified by the spool_id.

−purge

The *purge* option allows the user to delete a spooled file from the spooler subsystem. This command functions only for the

superuser or the originating user. The various forms of this command command are listed below:

spool −purge −sp spool-id

Purge the file specified by spool-id.

spool −purge −ws t01

Purge all files submitted from work station t01.

spool −purge −pt lp01

Purge all files queued to lp01.

spool −purge −who smith

Purge all files submitted by user smith.

−start

The *start* option starts printing on the specified printer. If the despooler is already writing to the printer, this command is ignored. The secondary option **−pt** indicates the printer. For example,

spool −start −pt lp01

starts printing with the first file on the spool queue for lp01.

−stop

The *stop* option stops printing on the specified printer and terminates the running despooler for the specified print device. The file which was being printed remains on the print queue and is resumed when an explicit *spool start* command is issued or a new print request is generated by the *print* or *lpr* command. The secondary option **−pt** indicates the printer. For example,

spool −stop −pt lp02

stops printing the file presently being printed on lp02.

−restart

The *restart* option restarts the printing of a file which is currently being printed at any page within the print file. If the specified page number is greater than the number of pages in the file, printing starts with page 1. The secondary option **−pt** indicates the printer. For example,

spool −restart −pt lp02

restarts printing at page 1 of the currently printing file on lp02.

The secondary option **−pg** indicates the page number. For example,

spool −restart −pt lp02 −pg 33

restarts printing at page 33 of the currently printing file on lp02.

−hold

The *hold* option allows a specific print file to be placed on hold by specifying the spool_id for the print file as follows:

spool −hold −sp spool_id

Also, a queue of files may be placed on hold by specifying the terminal work station, the print device, or the originating user name for the print file as shown in the following examples.

spool −hold −ws t02

Holds all files submitted from work station t02.

spool −hold −pt lp01

Holds all print files to be printed on lp01.

spool −hold −who smith

Holds all files originated by user smith.

−release

The *release* option allows a specific file to be released from hold by specifying the spool_id for the print file as follows:

spool −release −sp spool_id

Also, a group of print files may be released from hold by specifying the terminal work station, the destination print device, or the originator of the print file. The following examples define these options.

spool −release −ws t01

Releases all print files on hold for device t01.

spool −release −pt lp01

Releases all print files on hold for lp01.

spool −release −who smith

Releases all files on hold for user smith.

−look

The *look* option displays a spooled file to a terminal if the requesting user is the originator of the spooled file or the superuser. The secondary option **−sp** designates the spooled file.

FILES

/usr/spool/lpd/*	spool area
/usr/spool/lpd/lpd	despooler
/bin/print	spooler
/bin/lpr	spooler
/bin/spool	spool queue manager
/usr/spool/lpd/spooldev	spool device table manager
/usr/spool/lpd/??spldev	spool device table
/usr/spool/lpd/??splque	spool queue
/usr/spool/lpd/sf*	spooled files

SEE ALSO
print(1), spooldev(1M).

SUPPORT STATUS
Supported.

NAME

ssp − make output single spaced

SYNOPSIS

ssp [name ...]

DESCRIPTION

Ssp removes extra blank lines and causes all output to be single spaced. *Ssp* can be used directly, or as a filter after *nroff* or other text formatting operations.

EXAMPLE

```
nroff -ms filea fileb | ssp >> filec
```

prepares *filea* and *fileb* with the **−ms** macro package, then single spaces the output and directs it to *filec*.

SUPPORT STATUS

Supported.

NAME

stat − statistical network useful with graphical commands

SYNOPSIS

node-name [options] [files]

DESCRIPTION

Stat is a collection of command level functions (nodes) that can be interconnected using *sh*(1) to form a statistical network. The nodes reside in **/usr/bin/graf** (see *graphics*(1G)). Data is passed through the network as sequences of numbers (vectors), where a number is of the form:

[sign](digits)(.digits)[e[sign]digits]

evaluated in the usual way. Brackets and parentheses surround fields. All fields are optional, but at least one of the fields surrounded by parentheses must be present. Any character input to a node that is not part of a number is taken as a delimiter.

Stat nodes are divided into four classes.

Transformers,	which map input vector elements into output vector elements;
Summarizers,	which calculate statistics of a vector;
Translators,	which convert among formats; and
Generators,	which are sources of definable vectors.

Below is a list of synopses for *stat* nodes. Most nodes accept options indicated by a leading minus (−). In general, an option is specified by a character followed by a value, such as c5. This is interpreted as c := 5 (c is assigned 5). The following keys are used to designate the expected type of the value:

c	characters,
i	integer,
f	floating point or integer,
file	file name, and
string	string of characters, surrounded by quotes to include a *Shell* argument delimiter.

Options without keys are flags. All nodes except *generators* accept files as input, hence it is not indicated in the synopses.

Transformers:

abs	[−c*i*] − absolute value columns (similarly for −c options that follow)
af	[−c*i* t v] − arithmetic function titled output, verbose
ceil	[−c*i*] − round up to next integer
cusum	[−c*i*] − cumulative sum
exp	[−c*i*] − exponential

floor [−*ci*] − round down to next integer

gamma [−*ci*] − gamma

list [−*ci* d*string*] − list vector elements
delimiter(s)

log [−*ci* b*f*] − logarithm
base

mod [−*ci* m*f*] − modulus
modulus

pair [−*ci* F*file* x*i*] − pair elements
File containing base vector, x group size

power [−*ci* p*f*] − raise to a power
power

root [−*ci* r*f*] − take a root
root

round [−*ci* p*i* s*i*] − round to nearest integer, .5 rounds to 1
places after decimal point, significant digits

siline [−*ci* i*f* n*i* s*f*] − generate a line given slope and intercept
intercept, number of positive integers, slope

sin [−*ci*] − sine

subset [−a*f* b*f* *ci* F*file* *ii* l*f* **nl** **np** p*f* *si* *ti*] − generate a subset
above, below, File with master vector, interval, leave, master contains element numbers to leave, master contains element numbers to pick, pick, start, terminate

Summarizers:

bucket [−a*i* *ci* F*file* h*f* *ii* l*f* n*i*] − break into buckets
average size, File containing bucket boundaries, high, interval, low, number
Input data should be sorted

cor [−F*file*] − correlation coefficient
File containing base vector

hilo [− **h** **l** **o** **ox** **oy**]− find high and low values
high only, low only, option form, option form with x prepended, option form with y prepended

lreg [−F*file* **i** **o** **s**] − linear regression
File containing base vector, intercept only, option form for *siline*, slope only

mean [−f*f* n*i* p*f*] − (trimmed) arithmetic mean
fraction, number, percent

point [−f*f* n*i* p*f* **s**] − point from empirical cumulative density function
fraction, number, percent, sorted input

prod — internal product

qsort [—*ci*] — quick sort

rank — vector rank

total — sum total

var — variance

Translators:

bar [—**a b f g** *ri* **w***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] — build a bar chart
suppress axes, bold, suppress frame, suppress grid, region, width in percent, x origin, suppress x-axis label, y origin, suppress y-axis label, y-axis lower bound, y-axis high bound
Data is rounded off to integers.

hist [—**a b f g** *ri* **x***f* **xa y***f* **ya yl***f* **yh***f*] — build a histogram
suppress axes, bold, suppress frame, suppress grid, region, x origin, suppress x-axis label, y origin, suppress y-axis label, y-axis lower bound, y-axis high bound

label [—**b c F***file* **h p r***i* **x xu y yr**] — label the axis of a GPS file
bar chart input, retain case, label File, histogram input, plot input, rotation, x-axis, upper x-axis, y-axis, right y-axis

pie [—**b o p pn***i* **pp***i* **r***i* **v x***i* **y***i*] — build a pie chart
bold, values outside pie, value as percentage(:=100), value as percentage(:=i), draw percent of pie, region, no values, x origin, y origin
Unlike other nodes, input is lines of the form
[< **i e f** *cc* >] value [label]
ignore (do not draw) slice, explode slice, fill slice, color slice *c*=(black, red, green, blue)

plot [—**a b c***string* **d f F***file* **g m r***i* **x***f* **xa x***i* **xh***f* **xl***f* **xn***i* **xt y***f* **ya y***i* **yh***f* **yl***f* **yn***i* **yt**] — plot a graph
suppress axes, bold, plotting characters, disconnected, suppress frame, File containing x vector, suppress grid, mark points, region, x origin, suppress x-axis label, x interval, x high bound, x low bound, number of ticks on x-axis, suppress x-axis title, y origin, suppress y-axis label, y interval, y high bound, y low bound, number of ticks on y-axis, suppress y-axis title

title [—**b c l***string* **v***string* **u***string*] — title a vector or a GPS
title bold, retain case, lower title, upper title, vector title

Generators:

gas [−*ci* i*f* n*i* s*f* t*f*] − generate additive sequence
interval, number, start, terminate

prime [-*ci* h*i* l*i* n*i*] − generate prime numbers
high, low, number

rand [−*ci* h*f* l*f* m*f* n*i* s*i*] − generate random sequence
high, low, multiplier, number, seed

RESTRICTIONS

Some nodes have a limit on the size of the input vector.

SEE ALSO

graphics(1G), gps(4).

SUPPORT STATUS

Not supported.

NAME

strings − find the printable strings in a object, or other binary, file

SYNOPSIS

strings [−] [−**o**] [−*number*] file ...

DESCRIPTION

Strings looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null. *Strings* only looks in the initialized data space of object files.

Strings is useful for identifying random object files.

OPTIONS

− examine uninitialized data space as well as initialized data space

−**o** precede each string by its offset in the file (in octal)

−*n* use *n* as the minimum string length, rather than 4

SEE ALSO

od(1)

WARNING

The algorithm for identifying strings is extremely primitive

SUPPORT STATUS

Supported.

NAME

strip − strip symbol and line number information from object file

SYNOPSIS

strip [−l] [−x] [−r] [−V] filename

DESCRIPTION

The *strip* command strips the symbol table and line number information from common object files, including archives. Once a file has been *strip*ped, no symbolic debugging access is available for that file; therefore, *strip* is normally run only on production modules that have been debugged and tested. The purpose of this command is to reduce the file storage overhead taken by the object file.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *strip* prints an error message and terminates without stripping *filename* unless the −r option is used.

If the *strip* command is executed on a common archive file (see *ar*(4)) the archive symbol table is removed. The archive symbol table must be restored by executing the *ar*(1) command with the −s option before the archive can be link edited by the *ld*(1) command. *Strip*(1) instructs the user with appropriate warning messages when this situation arises.

OPTIONS

The amount of information stripped from the symbol table can be controlled by using any of the following options:

−l Strip line number information only; do not strip any symbol table information.
−x Do not strip static or external symbol information.
−r Reset the relocation indexes into the symbol table.
−V Print on the standard error output the version of the *strip* command which is executing.

NOTE

Release 2.x archive format is supported permitting transparent use of archive libraries from Release 2.x.

FILES

/usr/tmp/strp??????

SEE ALSO

ar(1), as(1), cc(1), ld(1), ar(4), a.out(4).

DIAGNOSTICS

If *filename* cannot be read, *strip* prints
strip: name: cannot open

If *filename* is not an appropriate common object file, *strip* prints
strip: name: bad magic

If *filename* contains relocation entries and the −r option is not used, the symbol table information cannot be stripped; *strip* prints
strip: name: relocation entries present; cannot strip

SUPPORT STATUS

Supported.

NAME

stty − set the options for a terminal

SYNOPSIS

stty [**−a**] [**−g**] [options]

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the **−a** option, it reports all of the option settings; with the **−g** option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below may be found in *termio*(7). Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (−parenb) enable (disable) parity generation and detection.

parodd (−parodd) select odd (even) parity.

cs5 cs6 cs7 cs8 select character size (see *termio*(7)).

0 hang up phone line immediately.

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)

hupcl (−hupcl) hang up (do not hang up) DATA-PHONE® connection on last close.

hup (−hup) same as **hupcl (−hupcl)**.

cstopb (−cstopb) use two (one) stop bits per character.

cread (−cread) enable (disable) the receiver.

clocal (−clocal) assume a line without (with) modem control.

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) upper-case alphabetics to lower case on input.

ixon (−ixon) enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (−ixany) allow any character (only DC1) to restart output.

ixoff (−ixoff)	request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.
Output Modes	
opost (−opost)	post-process output (do not post-process output; ignore all other output modes).
olcuc (−olcuc)	map (do not map) lower-case 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) upper/lower-case 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 erases 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**, **ctab**, **min**, or **time** (**ctab** is used with **−stappl**; (**min** and **time** are used with **−icanon**. 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; see *termio*(7).

term set all modes suitable for the terminal type *term*.

EXAMPLE

```
stty
```

to display options.

```
stty </dev/tty01
```

to display options for tty01.

SEE ALSO

tabs(1), ioctl(2), termio(7).

SUPPORT STATUS

Supported.

NAME

su − become superuser or another user

SYNOPSIS

su [−] [name [arg ...]]

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., superuser).

To use *su*, the appropriate password must be supplied (unless one is already **root**). If the password is correct, *su* executes a new shell with the real and effective user ID set to that of the specified user. The new shell is the optional program named in the shell field of the specified user 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 gives the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user password file entry is like *sh*(1). If the first argument to *su* is a −, the environment is 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 profile (**/etc/profile**) and then the specified user 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 **.profile** can check *arg0* for **−sh** or **−su** to determine if it was invoked by *login*(1) or *su*(1), respectively. If the user program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *−program* by both *login*(1) and *su*(1).

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:

```
su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
su − bin
```

To execute *command* with the temporary environment and permissions of user **bin**, execute:

```
su − bin −c "command args"
```

FILES

/etc/passwd	system password file
/etc/profile	system profile
$HOME/.profile	user profile
/usr/adm/sulog	log file

SEE ALSO

env(1), login(1), sh(1), passwd(4), profile(4), environ(5).

SUPPORT STATUS

Supported.

NAME

sum − print checksum and block count of a file

SYNOPSIS

sum [**−r**] file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. *Sum* is typically used to look for bad sections of a file, or to validate a file communicated over some transmission line.

OPTIONS

−r use an alternate algorithm in computing the checksum.

SEE ALSO

wc(1).

DIAGNOSTICS

Read error

means *end of file* for most devices; check the block count to determine if an actual read error occurred.

SUPPORT STATUS

Supported.

NAME

sync − update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive.

If the system is to be stopped, *sync* must be called to insure file system integrity. *Sync* flushes all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync*(2) for details.

SEE ALSO

sync(2).

SUPPORT STATUS

Supported.

NAME

tabs − set tabs on a terminal

SYNOPSIS

tabs [tabspec] [+mn] [−Ttype]

DESCRIPTION

Tabs sets the tab stops on the terminal according to the tab specification *tabspec*, after clearing any previous settings. The user terminal must have remotely-settable hardware tabs.

GE TermiNet terminals behave in a different way than most other terminals for some tab settings: the first number in a list of tab settings becomes the *left margin* on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 sets the left margin on a TermiNet, but not on other terminals. A tab list beginning with 1 has the same effect on all terminals.

Setting the left margin is possible on some other terminals (see below).

Tabs usually must know the type of terminal in order to set tabs; *tabs* always must know the terminal type in order to set margins. This type may be specified using the **−T** option (see below), but if no **−T** option is specified, *tabs* searches for the **$TERM** value in the *environment* (see *environ*(5)). If no type can be found, *tabs* tries a sequence that works for many terminals.

Tabs sets the tabs and margins using the standard output.

TAB SPECIFICATIONS

Tabs accepts four types of tab specification for *tabspec*: standard, repetitive, arbitrary, and file. If no *tabspec* is given, the default value is **−8**, (the UNIX system standard tabs) and 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.

The following *tabspecs* invoke standard tabs suited for particular languages:

−a 1,10,16,36,72
Assembler, IBM S/370, first format

−a2 1,10,16,40,72
Assembler, IBM S/370, second format

−c 1,8,12,16,20,55
COBOL, normal format

−c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:

<:t−c2 m6 s66 d:>

−c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **−c2**. **−c3** is the recommended format for COBOL. The

appropriate format specification is:

<:t−c3 m6 s66 d:>

−f 1,7,11,15,19,23
FORTRAN

−p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

−s 1,10,55
SNOBOL

−u 1,12,20,44
UNIVAC 1100 Assembler

In addition to these standard formats, three other types exist:

−*n* A repetitive specification requests tabs at columns 1+*n*, 1+2**n*, etc. Note that such a setting leaves a left margin of *n* columns on TermiNet terminals *only*. Of particular importance is the value −8: this represents the UNIX system standard tab setting, and is the most likely tab setting to be found at a terminal. This setting is required for use with the *nroff*(1) **−h** option for high-speed output. Another special case is the value −0, implying no tabs at all.

n1,n2,...
The arbitrary format permits the user to specify 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, the number is assumed to be an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

−−*file* If the name of a file is given, *tabs* reads the first line of the file. If *tabs* finds a format specification on the line, *tabs* sets the tab stops according to the specification; otherwise *tabs* sets the tabs as −8.

This specification may be used with the *pr*(1) command to assure that a tabbed file is printed with correct tab settings:

tabs −− file; pr file

OPTIONS

Any of the following may be used in addition to the tab specification; if a given option occurs more than once, the last value specified takes effect:

−T*type*
Denotes *type* as the terminal type, where *type* is a name listed in *term*(5).

+m*n* Moves all tabs 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 moves 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 option is given explicitly.

DIAGNOSTICS

illegal tabs	when arbitrary tabs are ordered incorrectly.
illegal increment	when *tabs* finds a zero or missing increment in an arbitrary specification.
unknown tab code	when a standard code cannot be found.
cannot open	when −−*file* option is used, and *tabs* cannot open the *file*.
file indirection	when −−*file* option is used and the specification in that file points to another file.

SEE ALSO

nroff(1), environ(5), term(5).

RESTRICTIONS

The ways of clearing tabs and setting the left margin are inconsistent among different terminals. Usually the left margin cannot be changed without also setting tabs.

Tabs clears only 20 tabs (on terminals requiring a long sequence), but sets 64.

SUPPORT STATUS

Supported.

NAME

tail − deliver the last part of a file

SYNOPSIS

tail [±[number][**lbc**[**f**]]] [file]

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance +*number* from the beginning, or −*number* from the end of the input (if *number* is null, the value +10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the −**f** (follow) option, if the input file is not a pipe, the program does not terminate after the line of the input file has been copied, but enters an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus *tail* may be used to monitor the growth of a file that is being written by some other process.

EXAMPLES

```
tail −f carter
```

prints the last ten lines of the file **carter**, followed by any lines that are appended to **carter** between the time *tail* is initiated and and when *tail* is killed. As another example, the command:

```
tail −15cf thompson
```

prints the last 15 characters of the file **thompson**, followed by any lines that are appended to **thompson** between the time *tail* is initiated and and when *tail* is killed.

SEE ALSO

dd(1).

RESTRICTIONS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length.

WARNING

Tail may produce irregular output when input includes character special files.

SUPPORT STATUS

Supported.

NAME

tar — tape file archiver

SYNOPSIS

tar [options] files

DESCRIPTION

Tar saves and restores files as though the files were on magnetic or streaming tape. Its actions are controlled by *options*. Note that *tar* normally functions silently.

Options designates a string of characters containing at most one function letter and possibly one or more function modifiers.

Other arguments to the command are *files* (or directory names) specifying which files are to be saved or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

FUNCTION LETTERS

- **r** Write the named files to the end of the tape. The **c** function implies this function. This option is valid only for magnetic reel tape.
- **x** Extract the named *files* from the tape. If a named file specifies a directory whose contents have been written onto the tape, *tar* recursively extracts this directory. If the named file on tape does not exist on the system, *tar* creates the file with the same mode as the one on tape except that the set-user-ID bit and the set-group-ID bit are not set unless you are the superuser. If the files exist, their modes are not changed except for the bits described above. *Tar* restores the owner, modification time, and mode (if possible). If no *files* are specified, *tar* extracts the entire content of the tape. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.
- **t** List the names of the specified files each time that they occur on the tape. If no *files* are specified, *tar* lists all the names on the tape.
- **u** Add the named files to the tape if they are not already there or have been modified since last written on that tape. This option is only valid for magnetic reel tape.
- **c** Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This function implies the **r** function.

FUNCTION MODIFIERS

The following characters may be used in addition to the letter that selects the desired function:

- **#s** Select the drive on which the tape is mounted and the density. # is the tape drive number (0-7) and s is the density: l — low (800 bpi), m — medium (1600 bpi), or h — high (6250 bpi). The default is **0m**.
- **v** Type the name of each file *tar* processes preceded by the function letter. When used with the **t** function, **v** gives more information about the tape entries than just the name.

w Print the action to be taken, followed by the name of the file, and then wait for the confirmation from the user. If a word beginning with **y** is entered, *tar* performs the action. Any other input cancels the action.

f archive
Use *archive* as the name of the archive instead of /dev/rstp/0yy. If *archive* is −, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *Tar* can also be used to move hierarchies with the command:

```
cd fromdir; tar cf − . | (cd todir; tar xf −)
```

b Use the next argument as the blocking factor for tape records. The default is 1; the maximum is 20. This option should only be used with raw archives (see **−f** above). The block size is determined automatically when reading tapes (function letters **x** and **t**).

l Print an error message if *tar* cannot resolve all of the links to the files being saved. If **l** is not specified, no error messages are printed.

m Do not restore the modification times. The modification time of the file is the time of extraction.

o Cause extracted files to take on the user and group identifier of the user running *tar* rather than those on tape.

EXAMPLE

To preserve ownership, modification date, and permissions over a *uucp*(1) communication, create an archive file and communicate it:

```
cd /usr/src/xxx
tar cf /tmp/xxx.tar .
uucp /tmp/xxx.tar tower1! username
```

To restore the archived files on tower1:

```
cd /usr/src/xxx
tar xvf /usr/spool/uucppublic/username/xxx.tar
```

FILES

/dev/rstp/*
/tmp/tar*

RESTRICTIONS

There is no way to ask for the *n*-th occurrence of a file.
Tar does not handle errors gracefully.
The **u** option can be slow.
The **b** option should not be used with archives on tape that are going to be updated. The **b** option should not be used with archives on disk because updating an archive on disk can destroy it.
The limit on file-name length is 100 characters.
Note that **tar c0m** is not the same as **tar cm0**.
The **−r** and **−u** options are valid only for magnetic reel tape. These options cause a read after seek error on streaming cartridge tape.

SUPPORT STATUS

Supported.

NAME

tbl − format tables for nroff or troff

SYNOPSIS

tbl [−TX] [files]

DESCRIPTION

Tbl is a preprocessor that formats tables for *nroff* or *troff*(1). *Tbl* copies the input files to the standard output, except for lines between .TS and .TE command lines, which are assumed to describe tables and are re-formatted by *tbl*. (The .TS and .TE command lines are not altered by *tbl*).

.TS is followed by global options. The global options, if any, are terminated with a semi-colon (;).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except the last format line, which describes *all* remaining lines of the table. The last format line must end with a period. Each column of each line of the table is described by a single format letter, optionally followed by format specifiers that determine the font and point size of the corresponding item, that indicate where vertical bars are to appear between columns, that determine column width, inter-column spacing, etc.

The format lines are followed by lines containing the actual data for the table, followed by .TE. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only _ or =, a single or double line, respectively, is drawn across the table at that point; if a *single item* in a data line consists of only _ or =, then that item is replaced by a single or double line.

The **−TX** option forces *tbl* to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., line printers).

If no file names are given as arguments (or if − is specified as the last argument), *tbl* reads the standard input, thus *tbl* may be used as a filter. When it is used with *eqn*(1) or *neqn*, *tbl* should come first to minimize the volume of data passed through pipes.

GLOBOL OPTIONS

center	center the table (default is left-adjust);
expand	make the table as wide as the current line length;
box	enclose the table in a box;
doublebox	enclose the table in a double box;
allbox	enclose each item of the table in a box;
tab (*x*)	use the character *x* instead of a tab to separate items in a line of input data.

FORMAT LETTERS

c	center item within the column;
r	right-adjust item within the column;
l	left-adjust item within the column;

- **n** numerically adjust item in the column: units positions of numbers are aligned vertically;
- **s** span previous item on the left into this column;
- **a** center longest line in this column and then left-adjust all other lines in this column with respect to that centered line;
- **^** span down previous entry in this column;
- **_** replace this entry with a horizontal line;
- **=** replace this entry with a double horizontal line.

FORMAT SPECIFIERS

- **B** Bold font
- *I* Italic font
- **|** Vertical line between columns

EXAMPLE

If → represent a tab (which should be typed as a genuine tab), then the input:

```
.TS
center box ;
cB s s
cI | cI s
^ | c c
l | n n .
Household Population
_
Town→Households
→Number→Size
=
Bedminster→789→3.26
Bernards Twp.→3087→3.74
Bernardsville→2018→3.30
Bound Brook→3425→3.04
Bridgewater→7897→3.81
Far Hills→240→3.19
.TE
```

yields:

Household Population		
Town	*Households*	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

SEE ALSO

cw(1), eqn(1), mm(1), mmt(1), nroff(1), mm(5), mv(5).

RESTRICTIONS

See *RESTRICTIONS* under *nroff*(1).

SUPPORT STATUS
Supported.

NAME

tc − phototypesetter simulator

SYNOPSIS

tc [−t] [−sn] [−pl] [file]

DESCRIPTION

Tc interprets its input (standard input default) as device codes for a Wang Laboratories, Inc. C/A/T phototypesetter. The standard output of *tc* is intended for a Tektronix 4014 terminal with ASCII and APL character sets. The sixteen typesetter sizes are mapped into the four sizes supported on the 4014 terminal; the entire TROFF character set is drawn using the character generator in the 4014, with overstruck combinations where necessary. Typical usage is:

troff −t files | tc

At the end of each page, *tc* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, three commands can be entered:

e *suppress* es the screen erase before the next page

s*n* skips the next *n* pages to be skipped

!*cmd*
sends *cmd* to the shell.

OPTIONS

−t Do not wait between pages (for directing output into a file).

−s*n* Skip the first *n* pages.

−p*l* Set page length to *l*; *l* may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); default is picas.

SEE ALSO

4014(1), sh(1), tplot(1G), troff(1).

RESTRICTIONS

Font distinctions are lost.

SUPPORT STATUS

Supported.

TEE(1)

NAME

tee − copy input to standard output and to files

SYNOPSIS

tee [**−i**] [**−a**] [file] ...

DESCRIPTION

Tee copies the standard input to the standard output and makes copies in the *files* overwriting their previous contents.

OPTIONS

−i Ignore interrupts.

−a Append the output to the *files* rather than overwriting them.

EXAMPLE

To print the *nroff*(1) format of *filex* and save the formatted file in *filex.nro*:

```
nroff filex|tee filex.nro|lpr
```

SUPPORT STATUS

Supported.

NAME

test − condition evaluation command

SYNOPSIS

test expr
[expr]

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, *test* returns non-zero (false) exit status; *test* also returns a non-zero exit status if there are no arguments.

Notice that all the operators and primitives are separate arguments to *test*.

PRIMITIVES

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**.

OPERATORS

The above 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. Parentheses are meaningful to the shell and, therefore, must be escaped.

SEE ALSO

find(1), sh(1).

WARNING

In the second form of the command (i.e., the one that uses [], rather than the word *test*), the square brackets must be delimited by blanks.

SUPPORT STATUS

Supported.

NAME

time − time a command

SYNOPSIS

time command

DESCRIPTION

Time executes *command*, then prints the time elapsed during *command*, the time spent in the system, and the time spent in execution of *command*.

Times are reported in seconds and are printed on standard error.

Time prints the times on standard error.

SEE ALSO

timex(1), times(2).

SUPPORT STATUS

Supported.

NAME

timex − time a command; report process data and system activity

SYNOPSIS

timex [options] command

DESCRIPTION

Timex executes the given *command*, then reports in seconds elapsed time, user time and system time spent in execution. Optionally, *timex* processes accounting data for the *command* and lists or summarizes all its children, and reports total system activity during the execution interval.

The output of *timex* is written on standard error.

OPTIONS

−p List process accounting records for *command* and all its children. Suboptions **f**, **h**, **k**, **m**, **r**, and **t** modify the data items reported, as defined in *acctcom*(1). The number of blocks read or written and the number of characters transferred are always reported.

−o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

−s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

SEE ALSO

acctcom(1), sar(1).

WARNING

Process records associated with *command* are selected from the accounting file **/usr/adm/pacct** by inference, since process genealogy is not available. *Timex* includes background processes having the same user-id, terminal-id, and execution time window.

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
```

SUPPORT STATUS

Supported.

NAME

toc − graphical table of contents routines

SYNOPSIS

dtoc [directory]
ttoc mm-file
vtoc [−**cdhnimsvn**] [TTOC file]

DESCRIPTION

All of the commands listed below reside in **/usr/bin/graf** (see *graphics*(1G)).

dtoc

Dtoc makes a textual table of contents, TTOC, of all subdirectories beginning at *directory* (*directory* defaults to .) with one entry per directory. The entry fields from left to right are level number, directory name, and the number of ordinary readable files in the directory. *Dtoc* is useful in making a visual display of all or parts of a file system.

The following makes a visual display of all the readable directories under /:

dtoc / | vtoc | td

ttoc

Ttoc translates the table of contents generated by the .TC macro of *mm*(1) to TTOC format. *Ttoc* assumes that *mm* file uses the .H family of macros for section headers. If no *mm-file* is given, the standard input is assumed.

vtoc

Vtoc produces a GPS describing a hierarchy chart from a TTOC. The output drawing consists of boxes containing text connected in a tree structure. If no *file* is given, the standard input is assumed. Each TTOC entry describes one box and has the form:

id [*line-weight,line-style*] "*text*" [*mark*]

Id is an alternating sequence of numbers and dots. The *id* specifies the position of the entry in the hierarchy. The *id* **0.** is the root of the tree.

Line-weight is one of the following:

n, normal-weight; or
m, medium-weight; or
b, bold-weight.

Line-style is one of the following:

so, solid-line;
do, dotted-line;
dd, dot-dash line;
da, dashed-line; or
ld, long-dashed

Text is a character string surrounded by quotes. The characters between the quotes become the contents of the box. To include a quote within a box escape it with a backslash.

Mark is a character string (surrounded by quotes if it contains spaces). Any included quotes or dots must be escaped. *Vtoc* puts the string above the top right corner of the box.

Entry example: 1.1 b,da "ABC" DEF

Entries may span more than one line by escaping the new-line (\new-line).

Comments are surrounded by the /*,*/ pair. They may appear anywhere in a TTOC.

Options:

c Use text as entered, (default is all upper case).

d Connect the boxes with diagonal lines.

h*n* Set horizontal interbox space to *n*% of box width.

i Suppress the box *id*.

m Suppress the box *mark*.

s Do not compact boxes horizontally.

v*n* Set vertical interbox space to *n*% of box height.

SEE ALSO

graphics(1G), gps(4).

SUPPORT STATUS

Not supported.

TOUCH(1)

NAME

touch − update access and modification times of a file

SYNOPSIS

touch [**−amc**] [mmddhhmm[yy]] files

DESCRIPTION

Touch updates the access and modification times of each *file*. If no time is specified (see *date*(1)) the current time is used. If a file does not exist, *touch* creates the file.

OPTIONS

−a Update only the access time.

−m Update only the modification time.

−c Do not create the file if it does not exist.

EXIT STATUS

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).

SUPPORT STATUS

Supported.

NAME

tpcvt − filter for old streaming tape format

SYNOPSIS

tpcvt [**−VB** *filename*]

DESCRIPTION

Tpcvt filters the data from a streaming tape and makes sure the data was not written to the tape by a previous streaming tape driver. *Tpcvt* reads from standard input and writes to standard output. The **VB** option causes *tpcvt* to read from the specified file.

The input data may be data from an old or new tape; *tpcvt* determines the source of the data and produces the correct output.

Tpcvt should be used via a pipe for receiving data from a tape and sending data to a destination program such as *cpio*(1). The **VB** option should be used when input consists of multi-volume tapes which were created with the **T** option of *cpio*. When **VB** is specified, *tpcvt* prompts for new volumes. After all volumes are processed, respond with an end-of-file (control−d) to the prompt for the next volume.

SEE ALSO

tp(4), cpio(1).

SUPPORT STATUS

Not supported.

NAME

tplot − graphics filters

SYNOPSIS

tplot [−Tterminal [−e raster]]

DESCRIPTION

Tplot reads plotting instructions (see *plot*(4)) from the standard input and in general produces, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter **$TERM** (see *environ*(5)) is used. Known *terminals* are:

300 DASI 300.
300S DASI 300s.
450 DASI 450.
4014 TEKTRONIX 4014.
ver Versatec D1200A. This version of *tplot* 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).

SUPPORT STATUS

Not supported.

NAME

tput − query terminfo database

SYNOPSIS

tput[−T type] capname

DESCRIPTION

Tput uses the *terminfo(4)* database to make terminal-dependent capabilities and information available to the shell. *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, 1 for FALSE), and does no output.

−T*type* Indicates the type of terminal. Normally this option is unnecessary, as the default is taken from the environment variable $TERM.

capname Indicates the attribute from the *terminfo* database. See *terminfo(4)*.

EXAMPLES

tput clear	Echo 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'	Set shell variable **bold** to stand-out mode sequence for current terminal. This might be followed by a prompt: **echo "${bold}Please type in your name: \c"**
tput hc	Set exit code to indicate if current terminal is a hardcopy terminal.

FILES

/etc/term/?/*	Terminal descriptor files
/usr/include/term.h	Definition files
/usr/include/curses.h	

DIAGNOSTICS

Tput prints error messages and returns the following error codes on error:

−1 Usage error.
−2 Bad terminal type.
−3 Bad capname.

In addition, if a *capname* is requested for a terminal that has no value for that *capname* (e.g., tput -T450 lines), −1 is printed.

SEE ALSO

stty(1), terminfo(4).

SUPPORT STATUS

Supported.

TR(1)

NAME

tr − translate characters

SYNOPSIS

tr [**−cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. *Tr* maps input characters found in *string1* into the corresponding characters of *string2*.

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 assumed to be a large number; 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.

OPTIONS

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.

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 newline.

```
tr −cs "[A−Z][a−z]" "[\012*]" <file1 >file2
```

SEE ALSO

ed(1), sh(1), ascii(5).

RESTRICTIONS

ASCII NUL may not be used in *string1* or *string2*; *tr* always deletes NUL from input.

SUPPORT STATUS

Supported.

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.

SUPPORT STATUS

Supported.

NAME

tsort − topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is assumed.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data

there is an odd number of fields in the input file.

RESTRICTIONS

Uses a quadratic algorithm for the typical use of ordering a library archive file.

SUPPORT STATUS

Supported.

NAME

tty − get the name of the terminal

SYNOPSIS

tty [−l] [−s]

DESCRIPTION

Tty prints the path name of the terminal.

OPTIONS

−l Print the synchronous line number to which the terminal is connected if it is on an active synchronous line.

−s Suppress printing of the path name; generate the exit code only.

EXIT STATUS

2 Invalid options were specified.

0 Standard input is a terminal.

1 Otherwise.

DIAGNOSTICS

not on an active synchronous line

The standard input is not a synchronous terminal and −l is specified.

not a tty

The standard input is not a terminal and −s is not specified.

SUPPORT STATUS

Supported.

NAME

ul − underline output for a terminal

SYNOPSIS

ul [**−i**] [**−t** *terminal*] [*file* ...]

DESCRIPTION

Ul reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM.

Ul reads the file */etc/termcap* to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, *ul* degenerates to *cat*(1). If the terminal cannot underline, underlining is ignored.

OPTIONS

−i Indicate underlining by a separate line containing appropriate dashes; this is useful when you want to look at the underlining which is present in an *nroff*(1) output stream on a CRT-terminal.

−t Use the terminal kind *terminal* instead of the kind specified in the environment.

SEE ALSO

man(1), nroff(1).

RESTRICTION

Nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

SUPPORT STATUS

Supported.

NAME

umask − set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

Umask sets the user file-creation mode mask 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 shell recognizes and executes *umask*.

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2).

SUPPORT STATUS

Supported.

NAME

uname − print name of current UNIX system

SYNOPSIS

uname [−amnrsv]

DESCRIPTION

Uname prints the current system name of the UNIX system on the standard output file. *Uname* is mainly useful to determine what system one is using.

OPTIONS

−a Print all information. This is the same as entering all options.
−m Print the machine hardware name.
−n Print the nodename (the nodename may be a name that the system is known by to a communication network).
−r Print the operating system release.
−s Print the system name (default).
−v Print the operating system version.

RESTRICTIONS

Uname may print 68000 when the system is a 68010 when **−m** or **−a** option is used.

SEE ALSO

uname(2).

SUPPORT STATUS

Supported.

NAME

unget − undo a previous get of an SCCS file

SYNOPSIS

unget [−rSID] [−**s**] [−**n**] files

DESCRIPTION

Unget undoes the effect of a get −e made prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of − is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

OPTIONS

−r*SID*

Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the new delta). The use of this option is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and was omitted on the command line.

−**s** Suppress the printout, on the standard output, of the *SID* of the intended delta.

−**n** Retain the *get* file which is normally removed from the current directory.

SEE ALSO

delta(1), get(1), help(1), sact(1).
Source Code Control System User Guide in the Support Tools Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

SUPPORT STATUS

Supported.

NAME

uniq − report repeated lines in a file

SYNOPSIS

uniq [−**udc** [+n] [−n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines, and removes the second and succeeding copies of repeated lines. *Uniq* writes the remaining lines 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).

OPTIONS

−**u** Output only lines not repeated in the original file.

−**d** Output one copy of every *repeated* line (no other lines).

−**c** Generate normal output, preceding each line with the number of times it occurred (supersedes −**u** and −**d**).

−*n* Ignore the first *n* fields together with any blanks before each for the comparison. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its adjacent fields.

+*n* Ignore the first *n* characters for the comparison. *Uniq* skips fields before characters.

SEE ALSO

comm(1), sort(1).

SUPPORT STATUS

Supported.

NAME

units − interactive conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

Units specifies a quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

Units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, along with 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.

Units recognizes **lb** , rather than **pound** as a unit of mass.

Compound names are run together, (e.g. **lightyear**).

British units that differ from their U.S. counterparts should be prefixed thus: **brgallon**.

For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

/usr/lib/unittab

SUPPORT STATUS

Supported.

NAME

uucp, uulog, uuname — UNIX system to UNIX system copy

SYNOPSIS

uucp [options] source-files destination-file

uulog [options]

uuname [options]

DESCRIPTION

Uucp

Uucp copies files named by the *source-file* arguments to the file named by the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names which *uucp* knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, and only to a destination in PUBDIR (see below). Care should be taken to insure that intermediate nodes in the route are set up to foward information.

The shell metacharacters ?, *, and [...] appearing in *path-name* are expanded on the appropriate system. In order to send files that begin with a dot (e.g., .profile) the files must be qualified with a dot. For example: .profile, .prof*, and .profil? are correct; whereas *prof* and ?profile are not correct.

Path names may be one of:

- a full path name;
- a path name preceded by ~*user* where *user* is a login name on the specified system and is replaced by that user login directory;
- a path name preceded by ~/*user* where *user* is a login name on the specified system and is replaced by that user directory under PUBDIR; or
- a file name or path name; *uucp* prefixes either with the current directory.

If the result is an erroneous path name for the remote system the copy fails. If the *destination-file* is a directory, the last part of the *source-file* name is used.

Uucp preserves the read, write, and execute permissions across the transmission (see *chmod*(2)).

Uucp associates a job number with each request. This job number can be used by *uustat*(1C) to obtain status information or terminate the job.

The environment variable **JOBNO** and the **−j** option of *uucp* are used to control the listing of the *uucp* job number on standard output. If the environment variable **JOBNO** is undefined or set to **OFF**, the job number is not listed (default). If *uucp* is then invoked with the **−j** option, the job number is listed. If the

environment variable **JOBNO** is set to **ON** and is exported, a job number is written to standard output each time *uucp* is invoked. In this case, the **−j** option supresses output of the job number. *Uucp* does not generate a job number for a strictly local transaction.

The following options are interpreted by *uucp*:

−c Use the source file when copying out rather than copying the file to the spool directory (default).
−C Copy the source file to the spool directory.
−d Make all necessary directories for the file copy (default).
−e*sys*
Send the *uucp* command to system *sys* to be executed there. This is successful only if the remote machine allows the *uucp* command to be executed by **/usr/lib/uucp/uuxqt**.
−f Do not make intermediate directories for the file copy.
−j Control writing of the *uucp* job number to standard output by changing the value of the environment variable **JOBNO**.
−m*file*
Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.
−n*user*
Notify *user* on the remote system that a file was sent.
−r Queue job but do not start the file transfer process. By default a file transfer process is started each time *uucp* is evoked.

Uulog
Uulog queries a summary log of *uucp* and *uux*(1C) transactions in the file **/usr/spool/uucp/LOGFILE**.

The options cause *uulog* to print logging information:

−s*sys*
Print information about work involving system *sys*. If *sys* is not specified, then logging information for all systems is printed.
−u*user*
Print information about work done for the specified *user*. If *user* is not specified then logging information for all users is printed.

Uuname
Uuname lists the *uucp* names of known systems. A description is printed for each system that has a line of information in **/usr/lib/uucp/L.sys** or **/usr/lib/uucp/ADMIN**. The format of ADMIN is: *sysname* tab *description* tab.

The following options are interpreted by *uuname*.

−l Return the local system name.
−v Print additional information about each system.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory for receiving and sending (PUBDIR)

/usr/lib/uucp/* other data and program files

SEE ALSO

mail(1), uustat(1), uux(1C), chmod(2).
UNIX System to UNIX System Copy in the Support Tools Guide.
UUCP Administration in the Superuser Guide.

NOTE

The *.../usr/spool/uucp/lck.sq (can't lock)* message which may appear in the system error log describes a very temporary condition which is resolved. No action is necessary.

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons, you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to **~nuucp** or just **~**).

RESTRICTIONS

All files received by *uucp* are owned by *uucp*.

The **−m** option works only sending files or receiving a single file. Receiving multiple files specified by special shell characters **?**, *****, **and [...]** does not activate the **−m** option.

The **−m** option does not work if all transactions are local or if *uucp* is executed remotely using the **−e** option.

The **−n** option functions only when the source and destination are not on the same machine.

Only the first six characters of a *system-name* are significant. Any excess characters are ignored.

SUPPORT STATUS

Supported.

NAME

uustat − uucp status inquiry and job control

SYNOPSIS

uustat [options]

DESCRIPTION

Uustat displays the status of, or cancels, previously specified *uucp*(1) commands, or provides general status on *uucp* connections to other systems.

OPTIONS

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user.

The following options are mutually exclusive; that is, only *one* of the following may be specified on the command line.

−c*hour* Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the superuser.

−j*jobn* Report the status of the *uucp* request *jobn*(job*number*). If **all** is used for *jobn*, *uustat* reports the status of all *uucp* requests. An argument must be supplied; if *jobn* is omitted, *uustat* prints the usage message and fails.

−k*jobn* Kill the *uucp* request whose job number is *jobn*. The specified *uucp* request must belong to the user issuing the *uustat* command unless the user is the superuser.

−m*mch* Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, *uustat* provides the status of all machines known to the local *uucp*.

−M*mch* Same as the −*m* option except print two times: the time that the last status was obtained and the time that the last successful transfer to that system occurred.

−r*jobn* Rejuvenate *jobn*. Set the modification time of *jobn* to the current time. This prevents *uuclean*(1M) from deleting the job until the modification time of the job reaches the limit imposed by *uuclean*.

The following options are not mutually exclusive:

−o*hour* Report the status of all *uucp* requests which are older than *hour* hours.

−O Report the *uucp* status using the octal status codes listed below. If this option is not specified, *uustat* prints the verbose description with each *uucp* request.

−q List the number of jobs and other control files queued for each machine and the time of the oldest and youngest file queued for each machine. If a lock file exists for that system, *uustat* lists the date of creation for that file.

−s*sys* Report the status of all *uucp* requests which communicate with remote system *sys*.

−u*user* Report the status of all *uucp* requests issued by *user*.

−y*hour* Report the status of all *uucp* requests which are younger than *hour* hours.

EXAMPLE

The command:

```
uustat −ucarter −stower −y72
```

prints the status of all *uucp* requests that were issued by user *carter* to communicate with system *tower* within the last 72 hours.

STATUS CODES

The meanings of the job request status are:

job-number user remote-system command-time status-time status

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

Octal	Status
000001	the copy failed, but the reason cannot be determined
000002	permission to access local file is denied
000004	permission to access remote file is denied
000010	bad *uucp* command is generated
000020	remote system cannot create temporary file
000040	cannot copy to remote directory
000100	cannot copy to local directory
000200	local system cannot create temporary file
000400	cannot execute *uucp*
001000	copy (partially) succeeded
002000	copy finished, job deleted
004000	job is queued
010000	job killed (incomplete)
020000	job killed (complete)

The meanings of the machine accessibility status are:

system-name time status

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

FILES

/usr/spool/uucp	spool directory
/usr/lib/uucp/L_stat	system status file
/usr/lib/uucp/R_stat	request status file

SEE ALSO

uucp(1C), uuclean(1M).

SUPPORT STATUS

Supported.

NAME

uuto, uupick − public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [options] source-files destination

uupick [−**s** system]

DESCRIPTION

Uuto

Uuto sends *source-files* to *destination*. *Uuto* uses the *uucp*(1C) facility to send files, while it allows the local system to control the file access.

A *source-file* name is a path name on your machine. In order to send files that begin with a dot (e.g., **.profile**) the files must be qualified with a dot. For example: **.profile**, **.prof***, and **.profil?** are correct; whereas ***prof*** and **?profile** are not correct.

Destination has the form:

system!*user*

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*(1)). *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.

Uuto sends the files (or sub-trees if directories are specified) to PUBDIR on *system*, where PUBDIR is a public directory defined to *uucp*. Specifically *uuto* sends the files to:

PUBDIR/receive/*user*/*mysystem*/files

Uuto notifies the destined recipient by *mail*(1) of the arrival of files.

Uupick

Uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, *uupick* prints the following message on the standard output:

from *system*: [file *file-name*] [dir *dirname*] ?

Uupick then reads a line from the standard input to determine the disposition of the file:

<new-line>	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir* (current directory is default).
a [*dir*]	Same as **m** except move *all* the files sent from *system*.

p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

Uupick invoked with the **−s** *system* option only searches the PUBDIR for files sent from *system*.

FILES

/usr/spool/uucppublic public directory (PUBDIR)

SEE ALSO

mail(1), uuclean(1M), uucp(1C), uustat(1C), uux(1C).

SUPPORT STATUS

Supported.

NAME

uux — UNIX-to-UNIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

Uux gathers zero or more files from various systems, executes a command on a specified system, and then sends standard output to a file on a specified system. Note that, for security reasons, many installations limit the list of commands executable on behalf of an incoming request from *uux*. Many sites permit little more than the receipt of mail (see *mail*(1)) via *uux*.

When *uux* gathers the files, the files are given permission 600 and are owned by *uucp*. The actual program running is *nuucp* so access permission to the files may be denied. For example, *print*(1) checks permissions and fails. Redirect standard input for *uux* to print the files.

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

- a full path name;
- a path name preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user login directory; or
- a file name or path name; *uux* prefixes either with the currect directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire *command-string* or quoting the special characters as individual arguments.

Uux attempts to get all files to the execution system. For files which are output files, the file name must be escaped using parentheses.

Uux notifies you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine. Executable commands are listed in **/usr/lib/uucp/L.cmds** on the remote system. The format of the **L.cmds** file is:

cmd,machine1,machine2,...

If no machines are specified, then any machine can execute **cmd**. If machines are specified, only the listed machines can execute **cmd**. If the desired command is not listed in **L.sys**, then no machine can execute that command.

Redirection of standard input and output is usually restricted to files in PUBDIR. Directories into which redirection is allowed must be specified in **/usr/lib/uucp/USERFILE** by the system administrator. The directory must have 777 permisions.

Uux associates a job number with each request. This job number can be used by *uustat*(1) to obtain status information or terminate

the job. The environment variable JOBNO and the −j option are used to control the listing of the *uux* job number on standard output. If the environment variable JOBNO is undefined or set to OFF, the job number is not listed (default). If *uux* is then invoked with the −j option, the job number is listed. If the environment variable JOBNO is set to ON and is exported, a job number is written to standard output each time *uux* is invoked. In this case, the −j option suppresses output of the job number.

OPTIONS

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

− The standard input to *uux* is made the standard input to the *command-string*.

−j Control writing of the *uux* job number to standard output by changing the value of the environment variable JOBNO.

−n Send no notification to user.

−m*file*
Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

−r Queue job but do not start the file transfer process. By default a file transfer process is started each time *uux* is evoked.

EXAMPLES

The command

```
uux "!diff tower1!/usr/carter/f1 tower2!/a4/carter/f1 > !f1.diff"
```

gets the **f1** files from the *tower1* and *tower2* machines, executes a *diff*(1) command and put the results in **f1.diff** in the local directory.

The command

```
uux tower1!uucp tower2!/usr/file \(tower3!/usr/file\)
```

sends a *uucp* command to system *tower1* to get **/usr/file** from system *tower2* and send it to system *tower3*.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory (PUBDIR)
/usr/lib/uucp/*	other data and programs

SEE ALSO

mail(1), uuclean(1M), uucp(1C).

RESTRICTIONS

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 * probably does not do what you want it to do. The shell tokens << and >> are not implemented.

Only the first six characters of the *system-name* are significant. Any excess characters are ignored.

To redirect output to a file, the name of the directory containing the file must be on the default system line in

/usr/lib/uucp/USERFILE. This allows any system to redirect to this directory.

SUPPORT STATUS
Supported.

NAME

val − validate SCCS file

SYNOPSIS

val −
val [−s] [−rSID] [−mname] [−ytype] files

DESCRIPTION

Val determines if the specified *file* is an SCCS file with the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of options, which begin with a −, and named files.

When the file argument − is specified, *val* reads the standard input until it detects an end-of-file condition. *Val* processes each input line independently as if the input was a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

OPTIONS

Each option affects each named file on the command line independently.

−s Silences the diagnostic message normally generated on the standard output for any error that *val* detects while processing each named file on a given command line.

−r*SID* Denotes the argument value *SID* (*S*CCS *ID*entification String) as an SCCS delta number. *Val* first determines whether the *SID* is ambiguous (e. g., r1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (e. g., r1.0 or r1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, *val* determines whether the corresponding file actually exists.

−m*name* Compares the *name* with the SCCS %M% keyword in *file*.

−y*type* Compares *type* with the SCCS %Y% keyword in *file*.

EXIT STATUS

The 8-bit code returned by *val* can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = invalid or ambiguous *SID*;
bit 5 = non-existent *SID*;
bit 6 = %Y%, −y mismatch;
bit 7 = %M%, −m mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases *val* returns an aggregate code: a logical OR of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), prs(1).

DIAGNOSTICS

Use *help*(1) for explanations.

RESTRICTIONS

Val can process up to 50 files on a single command line. Any number above 50 produces a core dump.

SUPPORT STATUS

Supported.

NAME

vc − version control

SYNOPSIS

vc [−a] [−t] [−cchar] [−s] [keyword=value ... keyword=value]

DESCRIPTION

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and any *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A *control statement* is a single line beginning with a control character, except as modified by the −t option (see below). The default control character is colon (:), except as modified by the −c option (see below). Input lines beginning with a backslash followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A *keyword* is composed of up to 9 alphanumerics, the first of which must be alphabetic. A *value* is any ASCII string that can be created with *ed*(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Vc replaces keywords with values whenever a keyword surrounded by control characters is encountered on a version control statement. The −a option (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with a backslash. If a literal backslash is desired, then it too must be preceded by backslash.

OPTIONS

−a	Replace keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.
−t	Ignore all characters from the beginning of a line up to and including the first *tab* character for the purpose of detecting a control statement. If a control statement is found, discard all characters up to and including the *tab*.
−c*char*	Use *char* as a control character in place of :.
−s	Silence warning messages (not error messages) that are normally printed on the diagnostic output.

VERSION CONTROL STATEMENTS

:dcl keyword[, ..., keyword]

Declares keywords. All keywords must be declared.

:asg keyword=value

Assigns values to keywords. An asg statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous asg's for that keyword. Keywords declared, but not assigned values have null values.

:if condition
:
:
:end

Skips lines of the standard input. If *condition* is true *vc* copies all lines between the *if* statement and the matching *end* statement to the standard output. If *condition* is false, *vc* discards all intervening lines, including control statements. Note that *vc* recognizes intervening *if* statements and matching *end* statements solely for the purpose of maintaining the proper *if-end* matching.

The syntax of a *condition* is:

```
<cond>      ::= [ "not" ] <or>
<or>        ::= <and>  <and> "" <or>
<and>       ::= <exp>  <exp> "&" <and>
<exp>       ::= "(" <or> ")"  <value> <op> <value>
<op>        ::= "="  "!="  "<"  ">"
<value>     ::= <arbitrary ASCII string>  <numeric
                string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (for example, 012 > 12 is false). All other operators take strings as arguments (for example : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

= != > < equal precedence
&

Parentheses may be used to alter the order of precedence. Values must be separated from operators or parentheses by at least one blank or tab.

::text
Replaces keywords on lines that are copied to the standard output. *Vc* removes the two leading control characters, and replaces keywords surrounded by control characters in text by their value before copying the line to the output file. This action is independent of the −a option.

:on

:off
Turn on or off keyword replacement on all lines.

:ctl char
Change the control character to char.

:msg message
Prints the given message on the diagnostic output.

:err message
Prints the given message followed by:
ERROR: err statement on line ... (915)
on the diagnostic output. *Vc* halts execution, and returns an exit code of 1.

DIAGNOSTICS
Use *help*(1) for explanations.

EXIT STATUS
0 − normal
1 − error

SUPPORT STATUS
Supported.

NAME

vi − screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [−t *tag*] [−r *file*] [−l] [−w*n*] [−x] [−R] [+*command*] name ...

view [−t *tag*] [−r *file*] [−l] [−w*n*] [−x] [−R] [+*command*] name ...

vedit [−t *tag*] [−r *file*] [−l] [−w*n*] [−x] [−R] [+*command*] name ...

DESCRIPTION

Vi (visual) is a display-oriented text editor based on the underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa. 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.

The *view* invocation is the same as *vi* except that the **readonly** flag is set preventing accidental overwriting of the file.

The *vedit* invocation is intended for beginners. The **report** flag is set to 1, and the **showmode** and **novice** flags are set. These defaults make it easier to get started learning the editor.

The following is summary information on *vi*. For a complete description and tutorial information on this editor, see the *Editing Guide*.

OPTIONS

The following invocation options are interpreted by *vi*:

−t *tag*
: Edit the file containing the *tag* and position the editor at its definition.

−r*file*
: Recover *file* after an editor or system crash. If *file* is not specified, a list of all saved files is printed.

−l
: Indent appropriately for lisp code, the () { } [[and]] commands in *vi* and **open** are modified to have meaning for *lisp* .

−w*n*
: Set the default window size to *n*. This is useful when using the editor over a slow speed line.

−x
: Set encryption mode; a key is prompted for allowing creation or editing of an encrypted file (see *crypt*(1)). This option is dependent on the encryption module being installed and linked on the system. Without the encryption module, text is unchanged.

−R
: Set read only mode; the **readonly** flag is set preventing accidental overwriting of the file.

+*command*
: The specified *ex* command is interpreted before editing begins.

name
: Files to be edited.

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 **a i A 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 interrupt.

Last Line
Reading input for : / ? or !; terminate with CR to execute, interrupt to cancel.

COMMAND SUMMARY

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text***ESC**	insert text *abc*
cw*new***ESC**	change word to *new*
easESC	pluralize word
x	delete a character
dw	delete a word
dd	delete a line
3dd	... 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
:*ex 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
^?	(delete or rubout) interrupts
^L	reprint screen if ^? scrambles it
^R	reprint screen if ^L is → key

File manipulation

:wCR	write back changes
: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 synonym for :e #
: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	to tag file entry *tag*
^]	:ta, following word is *tag*

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a CR.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
G	go to specified line (end default)
/*pat*	next line matching *pat*
?*pat*	prev line matching *pat*
n	repeat last / or ?
N	reverse last / or ?
/*pat*/+*n*	noth line after *pat*
?*pat*?−*n*	noth 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
^R	retype, eliminate @ lines
zCR	redraw, current at window top
z−CR	... at bottom
z.CR	... at center
/*pat*/z−CR	*pat* line at bottom
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
''	... at first non-white in line
m*x*	mark current position with letter *x*
`*x*	move cursor to mark *x*
'*x*	... at first non-white in line

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
0	beginning of line
$	end of line
h or →	forward
l or ←	backwards
^H	same as ←
space	same as →
f*x*	find *x* forward
F*x*	f backward
t*x*	upto *x* forward
T*x*	back upto *x*
;	repeat last f F t or T
,	inverse of ;
|	to specified column
%	find matching ({) or }

Words, sentences, paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(	back sentence
{	back paragraph
W	blank delimited word
B	back W
E	to end of W

Commands for LISP Mode

)	Forward s-expression
}	... but do not stop at atoms
(	Back s-expression
{	... but do not stop at atoms

Corrections during insert

^H	erase last character
^W	erase last word
erase	your erase, same as ^H
kill	your kill, erase input this line
\	quotes ^H, your erase and kill
ESC	ends insertion, back to command
^?	interrupt, terminates insert
^D	backtab over *autoindent*
↑^D	kill *autoindent*, save for next
0^D	... but at margin next also
^V	quote non-printing character

Insert and replace

a	append after cursor
i	insert before cursor
A	append at end of line

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
=	indent for LISP

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	... 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, the text in that buffer is put instead.

p	put back text after cursor
P	put 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

SEE ALSO

ex (1), crypt(1).
Visual Editor (vi) — Basic, *Visual editor (vi) — Advanced*, and *Text Editor (ex)* in the *Editing Guide*.

RESTRICTIONS

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.

VI(1)

SUPPORT STATUS

Supported. The *crypt* feature is available only in the United States.

NAME

wait — await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with & have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

RESTRICTIONS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

SUPPORT STATUS

Supported.

WC(1)

NAME

wc − word count

SYNOPSIS

wc [−lwc] [files]

DESCRIPTION

Wc counts lines, words, and characters in the named *files*, or in the standard input if no *files* are specified. It also keeps a total count for all named files.

A word is a string of characters delimited by spaces, tabs, or newlines.

When *files* are specified on the command line, they are printed along with the counts.

OPTIONS

The following options can be used in any combination:

−l Report number of lines only.
−w Report number of words only.
−c Report number of characters only.

SUPPORT STATUS

Supported.

NAME

what − identify SCCS files

SYNOPSIS

what [**−s**] files

DESCRIPTION

What searches the given *files* for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is @(#)) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file **f.c** contains

char ident[] = "@(#)identification information";

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

what f.c f.o a.out

prints

f.c: identification information

f.o: identification information

a.out: identification information

What is intended to be used in conjunction with the s-1SCCS command *get*(1), which automatically inserts identifying information, but *what* can also be used where the information is inserted manually.

OPTION

−s Quit after finding the first occurrence of pattern in each file.

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1.

Use *help*(1) for explanations.

RESTRICTIONS

An unintended occurrence of the pattern @(#) may be found just by chance, but this is harmless in most cases.

SUPPORT STATUS

Supported.

NAME

whereis − locate source, binary, and or manual for program

SYNOPSIS

whereis [**−sbm**] [**−u**] [**−SBM** dir ... **−f**] file ...

DESCRIPTION

Whereis locates source, binary, and manual sections for specified files.

Whereis first strips the supplied names of leading pathname components and any (single) trailing extension of the form *.ext*, e.g. .c. Prefixes of s. resulting from use of source code control are also stripped.

Whereis then attempts to locate the desired program in a list of standard places.

OPTIONS

One or two of the restrictive options, **−b**, **−s**, and **−m**, may be specified.

−b Search only for binary sections.

−s Search only for source sections.

−m Search only for manual sections.

−B, **−S**, and **−M**
Change or otherwise limit the places where *whereis* searches; *dir* must be a full pathname.

−f Terminate each directory list and signal the start of file names.

−u Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus **whereis −m −u *** asks for those files in the current directory which have no manual section.

EXAMPLE

The following example finds all the files in /usr/bin which are not documented in /usr/catman/u_man/man1 with source in /usr/src/cmd:

```
cd /usr/ucb
whereis −u −M /usr/catman/u_man/man1 −S /usr/src/cmd −f *
```

FILES

/usr/src/*
/usr/catman/*
/lib, /etc, /usr/{lib,bin,ucb}

RESTRICTIONS

Because *whereis* uses *chdir*(2) to run faster, pathnames given with the **−M**, **−S**, and **−B** must be full pathnames; that is, they must begin with a /.

SUPPORT STATUS

Supported.

NAME

who − who is on the system

SYNOPSIS

who [−abdHlpqrstTu] [file]

who am i

who am I

DESCRIPTION

Who lists the user name, terminal line, login time, elapsed time since activity occurred on the line, and the process identification of the command interpreter (shell) for each current UNIX system user. *Who* examines the **/etc/utmp** file to obtain its information. If *file* is given, that file is examined. Usually, *file* is **/etc/wtmp**, which contains a history of all the logins since the file was last created.

Who **am i** or *who* **am I** identifies the invoking user.

Except for the **−s** option (which is assumed if no options are specified), the general format for output entries is:

name [state] line time activity pid [comment] [exit]

OPTIONS

With options, *who* lists logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

−a Process **/etc/utmp** or the named *file* using all of the options.

−b Indicate the time and date of the last reboot.

−d Display 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 of the dead process as returned by *wait*(2). This option can be useful in determining why a process terminated.

−H Print column headings above the regular output.

−l List 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.

−p List any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from **/etc/inittab** that spawned this process. See *inittab*(4).

−q Display only the names and numbers of users currently logged on. When this option is used, all other options are ignored.

−r Indicate the current *run-level* of the *init* process.

−s List only the *name*, *line*, and *time* fields (default).

−t Indicate the last change to the system clock by the *date*(1) command by **root**.

−T Print the *state* of the terminal line as well as all fields requested by the **−u** option. The *state* describes whether someone else can write to that terminal. A + indicates the

terminal is writable by anyone; a − indicates 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.

−u List the following information about those users who are currently logged in:

- *name*− User login name.
- *line*− Line name as found in the directory **/dev**.
- *time*− Time that the user logged in.
- *activity*− Number of hours and minutes since activity last occurred on this particular line. A dot (.) indicates that the terminal has seen activity in the last minute. If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked **old**. This information is useful when trying to determine if a user is working at the terminal.
- *pid*− Process identification of the user shell.
- *comment*− Comment field associated with this line as found in **/etc/inittab** (see *inittab*(4)). This comment may contain the location of the terminal, the telephone number of the dataset, the type of terminal if hard-wired, etc.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), init(1M), login(1), mesg(1), wait(2), inittab(4), utmp(4).

SUPPORT STATUS

Supported.

NAME

write − write to another user

SYNOPSIS

write user [line]

DESCRIPTION

Write copies lines from your terminal to the terminal of another user. When first called, *write* sends the message

Message from *yourname* **(tty??)** [*date*]...

to the person you want to talk to. When the connection is completed, two bells are sent to your own terminal. Enter your message and press the newline or return key. Pressing the newline or return key at the end of entering your message sends the message. Each time you enter a message and press the newline key, *write* sends the message. The recipient of the message should use *write* to respond to your message.

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, for example **(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.

Communication continues until you enter an end of file (control-d) from the terminal or until *write* receives an interrupt or the recipient issues a *mesg*(1) command to deny messages. 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 is to receive your message, for example, **tty00**. Otherwise, the first instance of the user found in **/etc/utmp** is assumed and the following message is displayed:

User is logged on more than one place.
You are connected to *terminal.*
Other locations are: *terminal*

Permission to write may be denied or granted by use of the *mesg(1)* command. Writing to others is normally allowed by default. Certain commands, in particular *nroff*(1) and *pr*(1) disallow messages in order to prevent interference with their output. However, if the user has superuser permissions, *write* forces messages 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.

FILES

/etc/utmp to find user
/bin/sh to execute !

SEE ALSO

mail(1), mesg(1), nroff(1), pr(1), sh(1), who(1).

DIAGNOSTICS

User not logged on

The person you are trying to *write* to is not logged in. Use *who*(1) to determine who is logged in or *mailx*(1) to mail a message to the user who is not logged in.

Permission denied

The person you are trying to *write* to denies that permission with *mesg*(1). Try using *mailx*(1).

Warning: cannot respond, set mesg y

Your terminal is set to **mesg n** and the recipient cannot respond to you. Enter **!mesg y** to accept messages.

Can no longer write to user

The recipient denied permission (**mesg n**) after you started writing.

SUPPORT STATUS

Supported.

NAME

xargs − construct argument list(s) and execute command

SYNOPSIS

xargs [options] [command [initial-arguments]]

DESCRIPTION

Xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the options specified.

Xargs searches for *command*, which may be a shell file, using the **$PATH** of the user. If *command* is omitted, *xargs* uses **/bin/echo**.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; *xargs* discards empty lines. 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. A backslash escapes the next character if the backslash does not appear in a quoted string.

Xargs constructs each argument list starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **−i** option). Options **−i**, **−l**, and **−n** determine how arguments are selected for each command invocation. When none of these options are specified, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, then *xargs* executes *command* with the accumulated args. This process is repeated until there are no more args.

Xargs terminates if 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**.

OPTIONS

When there are option conflicts (e.g., **−l** vs. **−n**), the last option has precedence.

−l*number* execute *command* for each non-empty *number* lines of arguments from standard input.

The last invocation of *command* will have 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 or tab signals continuation through the next non-empty line.

If *number* is omitted, 1 is assumed.

Option **−l** forces option **−x**.

−i*replstr* (Insert mode) Execute *command* for each line from standard input, taking the entire line as a single arg, and 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*.

Xargs strips blanks and tabs from the beginning of each line.

Constructed arguments may become at most 255 characters long.

Option −i forces option −x.

{ } 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 are used if their total size is greater than *size* characters, and if there are fewer than *number* arguments remaining on the last invocation.

If option −x is also specified, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

−t (Trace mode) Echo the *command* and each constructed argument list to file descriptor 2 just prior to their execution.

−p (Prompt mode) Prompt the user whether to execute *command* before each invocation.

Trace mode (−t) prints the command instance to be executed, followed by a ?... prompt. A reply of y (followed by anything) executes the command; any other response, including pressing the carriage return, skips that particular invocation of *command*.

−x Terminate *xargs* if any argument list would be greater than *size* characters;

−x is forced by the options −i and −l. When neither of the options −i, −l, or −n are coded, the total length of all arguments must be within the *size* limit.

−s*size* Set the maximum total size of each argument list to *size* characters; *size* must be a positive integer less than or equal to 470. If −s is not coded, 470 is assumed.

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* Designate *eofstr* is taken as the logical end-of-file string.

Underscore (_) is assumed for the logical EOF string if —e is not coded.

—e with no *eofstr* coded turns off the logical EOF string capability (underscore is taken literally).

Xargs reads standard input until either end-of-file or the logical EOF string is encountered.

EXAMPLES

This example moves all files from directory $1 to directory $2, and echos each move command just before doing it:

```
ls $1 | xargs −i −t mv $1/{} $2/{}
```

This example combines the output of the parenthesized commands onto one line, which is then appended to the file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

This example asks the user 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
```

This example executes *diff*(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs −n2 diff
```

DIAGNOSTICS

Self explanatory.

SUPPORT STATUS

Supported.

NAME

xstr − extract strings from C programs to implement shared strings

SYNOPSIS

xstr [−c] [−] [file]

DESCRIPTION

Xstr maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only. *Xstr* reads from its standard input when the argument '−' is given.

The command

xstr −c name

extracts the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c*, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not change the data base.

After all components of a large program have been compiled, a file *xs.c* declaring the common *xstr* space can be created by a command of the form

xstr

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared), saving space and swap overhead.

Xstr can also be used on a single file. A command

xstr name

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc −E name.c | xstr −c −
cc −c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added, thus *make* can avoid remaking *xs.o* unless truly necessary.

FILES

strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array *xstr*
/tmp/xs*	Temp file when *xstr name* does not touch *strings*

SEE ALSO

mkstr(1)

WARNING

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings are placed in the data base, when just placing the longer one there suffices.

SUPPORT STATUS

Supported.

NAME

yacc − yet another compiler-compiler

SYNOPSIS

yacc [**−vdlt**] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; *yacc* uses specified precedence rules to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*.

Yacc always generates runtime debugging code in *y.tab.c* under conditional compilation control. This code is normally not included when **y.tab.c** is compiled. See the **−t** option below. The runtime debugging code is under the control of **YYDERESTRICTION**, a preprocessor symbol. If **YYDERESTRICTION** has a non-zero value, then the debugging code is included. If its value is zero, then the code is not be included. The size and execution time of a program produced without the runtime debugging code is smaller and slightly faster.

OPTIONS

−v Prepare the file **y.output**, which describes the parsing tables reports conflicts generated by ambiguities in the grammar.

−d Generate the file **y.tab.h** with the **#define** statements that associate the *yacc*-assigned token codes with the user-declared token names. This option allows source files other than **y.tab.c** to access the token codes.

−l Produce code in **y.tab.c** which does *not* contain any **#line** constructs. This option should only be used after the grammar and the associated actions are fully debugged.

−t Include runtime debugging code when **y.tab.c** is compiled.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.debug, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1).

Yet Another Compiler Compiler (yacc) in the Support Tools Guide.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, *yacc* also reports any rules that could not

be reached from the start symbol in **y.output**.

RESTRICTIONS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

SUPPORT STATUS

Supported.

INTRO(6)

NAME

intro — introduction to games

DESCRIPTION

This section describes the recreational and educational programs found in the directory **/usr/games**.

SUPPORT STATUS

Not supported.

NAME

arithmetic − provide drill in arithmetic problems

SYNOPSIS

/usr/games/arithmetic [+−x/] [range]

DESCRIPTION

Arithmetic types out simple arithmetic problems, and waits for an answer to be entered.

If the answer is correct, it replies

Right!

and prints a new problem.

If the answer is wrong, it replies

What?

and waits for another answer.

Every twenty problems, *arithmetic* publishes statistics on correctness and the time required to answer.

To quit the program, enter an interrupt (delete).

The program does not give correct answers, since the learner should, in principle, be able to calculate them.

For almost all users, the relevant statistic should be time per problem, not percent correct.

ARGUMENTS

One or more of the following characters specifies the type of problem to be generated. If more than one is given, the different types of probelms are mixed in random order; default is +−.

+	generate addition problems
−	generate subtraction problems
x	generate multiplication problems
/	generate division problems

Range is a decimal number; all operands and answers are less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

SUPPORT STATUS

Not supported.

NAME

back − the game of backgammon

SYNOPSIS

/usr/games/back

DESCRIPTION

Back is a program which provides a partner for the game of backgammon. It is designed to play at three different levels of skill, one of which you must select.

In addition to selecting the level of the opponent, you may also indicate that you would like to roll your own dice during your turns.

You are also be given the opportunity to move first. The practice of each player rolling one die for the first move is not incorporated.

The points are numbered 1−24: 1 is the extreme inner table of white, 24 is the inner table of brown, 0 is the bar for removed white pieces, and 25 is the bar for brown.

For details on how moves are expressed, type y when *back* asks

Instructions?

at the beginning of the game. When *back* first asks

Move?

type a question mark to see a list of move options other than entering your numerical move.

When the game is finished, *back* asks you if you want the log. If you respond with y, *back* attempts to append to or create a file **back.log** in the current directory.

FILES

/usr/games/lib/backrules	rules file
/tmp/b*	log temp file
back.log	log file

RESTRICTIONS

Back complains loudly if you attempt to make too *many* moves in a turn, but becomes very silent if you make too *few*.
Doubling is not implemented.

SUPPORT STATUS

Not supported.

NAME

bj – the game of black jack

SYNOPSIS

/usr/games/bj

DESCRIPTION

Bj is a serious attempt at simulating the dealer in the game of black jack (or twenty-one). The following rules apply:

The bet is $2 every hand.

A player *natural* (black jack) pays $3. A dealer natural loses $2. Both dealer and player naturals is a *push* (no money exchange).

If the dealer has an ace up, the player is allowed to make an *insurance* bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where the player wins $2 if the dealer has a natural and loses $1 if the dealer does not.

If the player is dealt two cards of the same value, he is allowed to *double*. He is allowed to play two hands, each with one of these cards. (The bet is doubled also; $2 on each hand.)

If a dealt hand has a total of ten or eleven, the player may *double down*. He may double the bet ($2 to $4) and receive exactly one more card on that hand.

Under normal play, the player may *hit* (draw a card) as long as his total is not over twenty-one. If the player *busts* (goes over twenty-one), the dealer wins the bet.

When the player *stands* (decides not to hit), the dealer hits until he attains a total of seventeen or more. If the dealer *Rbusts* , the player wins the bet.

If both player and dealer *stand*, the one with the largest total wins. A tie is a *push*.

The machine deals and keeps score. The following questions are asked at appropriate times. Each question is answered by y followed by a new-line for *yes*, or just new-line for *no*.

? (do you want a hit?)
Insurance?
Double down?

Every time the deck is shuffled, the dealer so states and the *action* (total bet) and *standing* (total won or lost) is printed.

To exit, hit the interrupt key (DEL); *bj* prints the action and standing.

SUPPORT STATUS

Not supported.

NAME

craps – the game of craps

SYNOPSIS

/usr/games/craps

DESCRIPTION

Craps is a form of the game of craps that is played in Las Vegas. The program simulates the *roller*, while the user (the *player*) places bets. The player may choose, at any time, to bet with the roller or with the *House*. A bet of a negative amount is taken as a bet with the House, any other bet is a bet with the roller.

The player starts off with a *bankroll* of $2,000.

The program prompts with:

bet?

The bet can be all or part of the bankroll of the player. Any bet over the total bankroll is rejected and the program prompts with **bet?** until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (the player wins or loses depending on whether the bet is placed with the roller or with the House; the odds are even). The *first* roll is the roll immediately following a bet:

1. On the first roll:

7 or 11	wins for the roller;
2, 3, or 12	wins for the House;
any other number	is the *point*, roll again (Rule 2 applies).

2. On subsequent rolls:

point	roller wins;
7	House wins;
any other number	roll again.

If a player loses the entire bankroll, the House offers to lend the player an additional $2,000. The program prompts:

marker?

A yes (or y) consummates the loan. Any other reply terminates the game.

If a player owes the House money, the House reminds the player, before a bet is placed, how many markers are outstanding.

If, at any time, the bankroll of a player who has outstanding markers exceeds $2,000, the House asks:

Repay marker?

A reply of yes (or y) indicates the willingness of the player to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker are outstanding, the House asks:

How many?

markers the player would like to repay. If an invalid number is entered (or just a carriage return), an appropriate message is

CRAPS(6)

printed and the program prompts with **How many?** until a valid number is entered.

If a player accumulates 10 markers (a total of $20,000 borrowed from the House), the program informs the player of the situation and exits.

Should the bankroll of a player who has outstanding markers exceed $50,000, the *total* amount of money borrowed is *automatically* repaid to the House.

Any player who accumulates $100,000 or more breaks the bank. The program then prompts:

New game?

to give the House a chance to win back its money.

Any reply other than **yes** is considered to be a **no** (except in the case of **bet?** or **How many?**). To exit, send an interrupt (break), DEL, or control-D. The program indicates whether the player won, lost, or broke even.

NOTES

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

SUPPORT STATUS

Not supported.

NAME

maze — generate a maze

SYNOPSIS

/usr/games/maze

DESCRIPTION

Maze asks a few questions and then prints a maze.

RESTRICTIONS

Some mazes (especially small ones) have no solutions.

SUPPORT STATUS

Not supported.

NAME

moo – guessing game

SYNOPSIS

/usr/games/moo

DESCRIPTION

Moo is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. The player guesses four distinct digits being scored on each guess.

A *cow* is a correct digit in an incorrect position.

A *bull* is a correct digit in a correct position.

The game continues until the player guesses the number (a score of four bulls).

SUPPORT STATUS

Not supported.

NAME

ttt – tic-tac-toe

SYNOPSIS

/usr/games/ttt

DESCRIPTION

Ttt is the X and O game popular in the first grade. This is a learning program that never makes the same mistake twice.

Although it learns, it learns slowly. It must lose nearly 80 games to completely know the game.

FILES

/usr/games/ttt.k learning file

SUPPORT STATUS

Not supported.

NAME

wump — the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

Wump plays the game of *Hunt the Wumpus.*

A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it gives a more detailed description if you want.

SUPPORT STATUS

Not supported.

SEE REVERSE SIDE OF THIS FORM FOR INSTRUCTIONS

DOCUMENTATION ORDER FORM

NCR Customer No.

Purchase Order No.	Purchase Order Date

SHIP TO: Ship To No.

Company Name

Address

City/State/Zip

Attention

In case we have questions regarding your order
Area Code ( ) Number – Ext.

☐ Change of Address

BILL TO: Bill To No.

Company Name

Address

City/State/Zip

Attention

In case we have questions regarding your order
Area Code ( ) Number – Ext.

☐ Change of Address

Ship VIA

International Only – Customs Declaration

Comments:

DOCUMENT NUMBER	QUANTITY	DESCRIPTION	UNIT PRICE	AMOUNT
			Subtotal	
			State/Local Taxes	
			Total	

Tax Exempt? ☐ No ☐ Yes | Tax Exempt No.

For your convenience you can:

1. Mail this order to:
   NCR Corporation
   Order Processing
   Publication Services
   Dayton, Ohio 45479
2. Submit this order to your local NCR office
3. Call our toll-free number:
   1-800-543-2010
   In Ohio:
   1-800-543-6691
   8:00 A.M.-4:30 P.M.
   EST - Weekdays

NCR

Signature

Date

ORDERING INFORMATION

HOW TO ORDER

For a fast, easy way to order and receive the documents you need, complete this Documentation Order Form as follows.

1. NCR CUSTOMER NO.

Enter your 8-digit NCR customer number.

CUSTOMER NUMBERS ARE REQUIRED ON ALL ORDERS.

If you are unsure of your customer number, please contact your local NCR office.

2. PURCHASE ORDER NO.

Enter your purchase order number.

3. PURCHASE ORDER DATE

Enter the purchase order date.

4. SHIP TO

a. If you wish to have documents shipped to a location different than the location associated with your NCR customer number, enter the appropriate information in the space provided.

b. If the order is to be shipped to your NCR customer number location, no entry is required in this space.

5. BILL TO

No entry is required unless the BILL TO location is different from the SHIP TO location.

6. SHIP VIA

Enter your preferred method of delivery. All orders for items in stock will be processed and shipped within one week of receipt of order via UPS or mail for domestic shipment and air shipment for international orders. Rush orders will be processed and delivered within 48 hours.

7. CUSTOMS DECLARATION
(For International Orders)

If a customs declaration is required, enter the full text of the declaration in the space provided.

8. COMMENTS

Use this space for special comments about your order.

9. DOCUMENT NUMBER

Enter the number of the document you wish to order. NCR document numbers have a 2-character prefix, followed by 5 to 7 characters. (D1-0000-00).

10. QUANTITY

Enter the quantity desired. The following discounts apply when any document is purchased in quantities of 10 or more on one order.

- 10-49 10% discount
- 50-99 15% discount
- 100 or more 20% discount

11. DESCRIPTION

Enter the title of the document.

12. UNIT PRICE

Enter the unit price of the document. (Prices listed are those in effect at the time of printing and are subject to change without notice.) All prices are quoted in U.S. dollars.

13. AMOUNT

To calculate the line item amount, multiply the quantity by the unit price.

14. SUBTOTAL

Add all entries in the amount column and enter the sum in the SUBTOTAL space.

15. STATE/LOCAL TAXES

Calculate applicable state and local taxes by multiplying SUBTOTAL amount by the percentage of tax. Enter that number in the STATE/LOCAL TAXES space. If tax exempt, your tax exempt number must be entered in the space provided to the left.

16. TOTAL

Enter the total amount of the order.

THREE CONVENIENT WAYS TO ORDER

Enter your order using one of the following convenient methods:

U.S. Customers

- Mail the Documentation Order Form to NCR Corporation, Order Processing—Publication Services, Dayton, Ohio 45479.
- Submit the order to your local NCR office.
- Call our toll-free number: **1-800-543-2010**; in Ohio, **1-800-543-6691.** Phone orders are taken from 8:00 am to 4:30 pm EST—weekdays.

International Customers

- Customers located outside of the United States should contact their local NCR office for ordering and pricing information.

PAYMENT METHODS

Payments are accepted by check, money order, or purchase order. Please do not send cash.

RETURNS

If you wish to return documents for credit, please contact our Order Processing department (1-800-543-2010; in Ohio, 1-800-543-6691) within 30 days of the shipment date. Full credit will be given if documents are returned because of an NCR error. A credit of up to 75% of the net amount may be issued for the return of unused, shrink-wrapped documents. Returns will NOT be accepted without prior approval from Publication Services.

OUT OF STOCK ITEMS

Every attempt will be made to ensure that your order is filled completely and accurately. If a document is temporarily out of stock, it will be placed on back order. When a partial shipment is made, your packing list will include a notification of this condition. Back orders will be automatically filled when the document is returned to our inventory.

Retain a copy of this order for your records. Thank you.

NCR

READER'S COMMENTS FORM

x 2066 90 0884

BOOK TITLE	BOOK NO.	PRINT DATE

To help us plan future editions of this document, please take a few minutes to answer the following questions. Explain in detail using the space provided. Include page numbers where applicable.

Are there any technical errors or misrepresentations in the document?

Is the material presented in a logical and consistent order?

Is it easy to locate specific information in the document?

Is there any information you would like to have added to the document?

Are the examples relevant to the task being described?

Could parts of the document be deleted without affecting the document's usefulness?

Did the document help you to perform your job?

Any general comments?

NAME _____

TITLE _____

COMPANY _____

ADDRESS _____

TELEPHONE NO. ( ) _____

Thank you for your evaluation of this document
Fold the form as indicated and mail to NCR No postage is necessary in the U S A

TAPE

fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO 3 DAYTON, OHIO

POSTAGE WILL BE PAID BY ADDRESSEE

NCR Corporation

ATTENTION: Publication Services
WHQ-4

Dayton, Ohio 45409