ALTOS

UNIX™ System V

User Reference

UNIX™ System V User Reference

ACKNOWLEDGEMENTS

The Altos logo, as it appears in this manual, is a registered trademark of Altos Computer Systems.

DIABLO™ is a trademark of Xerox Corporation.

HP™ is a trademark of Hewlett-Packard, Inc.

TEKTRONIX® is a registered trademark of Tektronics, Inc.

Versatec® is a registered trademark of Versatec Corporation.

TELETYPE™ is a trademark of AT&T Teletype Corporation.

UNIX™ is a trademark of AT&T Bell Laboratories.

WorkNet™ is a trademark of Altos Computer Systems.

3B™ and DOCUMENTER'S WORKBENCH™ are trademarks of AT&T Technologies.

INTRODUCTION

This manual describes the features of the UNIX system. It provides neither a general overview of the UNIX system nor details of the implementation of the system.

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

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

1. Commands and Application Programs:
   1. General-Purpose Commands
   1C. Communications 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 the user's programs. Commands generally reside in the directory **/bin** (for **bin**ary programs). Some programs also reside in **/usr/bin**, to save space in **/bin**. These directories are searched automatically by the command interpreter called the *shell*. Sub-class 1C contains communication programs such as *cu*, *send*, *uucp*, etc. These entries may not apply from system to system depending upon the hardware included on your processor. Some UNIX systems may have a directory called **/usr/lbin**, containing local commands.

Section 6 (*Games*) describes the games and educational programs that, as a rule, 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. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 *UNIX System User Guide* for a more complete introduction to the system.)

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

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

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

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

Logging out. There are two ways to log out:

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

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

UNIX 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 will have interspersed in it the input characters. However, whatever you type will be saved and interpreted in the correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system silently throws away *all* the saved characters.

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

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

The ASCII **DEL** (a.k.a. "rubout") character is not passed to programs, but instead generates an *interrupt signal,* just like the "break", "interrupt", or "attention" signal. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you 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, if possible, 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 will rescue you.

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

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

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

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

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

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

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

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

For a fuller discussion of the file system, see the references cited at the beginning of the *INTRODUCTION* above. It may also be useful to glance through Section 2 of The UNIX System Programmer Reference Manual, which discusses system calls, even if you do not intend to deal with the system at that level.

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

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

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

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

Text processing. Almost all text is entered through the editor *ed*(1). The commands most often used to write text on a terminal are *cat*(1), *pr*(1), and *nroff*. The *cat*(1) command simply dumps ASCII text on the terminal, with no processing at all. The *pr*(1) command paginates the text, supplies headings, and has a facility for multicolumn output.

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

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

TABLE OF CONTENTS

interface.	acu: Automatic Call Unit (ACU)	acu(7)
acu: Automatic Call Unit	(ACU) interface.	acu(7)
	adb: absolute debugger.	adb(1)
addhd:shell script for	add-on disk drive	addhd(1M)
acctmerg: merge or	add total accounting files.	acctmerg(1M)
putenv: change or	add value to environment.	putenv(3C)
SCCS files.	admin: create and administer	admin(1)
admin: create and	administer SCCS files.	admin(1)
of complex argument.	aimag: Fortran imaginary part	aimag(3F)
intrinsic function.	aint: Fortran integer part	aint(3F)
alarm: set a process	alarm clock	alarm(2)
clock.	alarm: set a process alarm	alarm(2)
brk: change data segment space	allocation.	brk(2)
malloc: main memory	allocator.	malloc(3C)
malloc: fastmain memory	allocator.	malloc(3X)
accept:	allow/prevent LP requests.	accept(1M)
boolean functions.	and, or, xor, not: Fortran	bool(3F)
sort: sort	and/or merge files.	sort(1)
send: gather files	and/or submit RJE jobs:	send(1C)
functions.	anint: Fortran nearest integer	round(3F)
link editor output.	a.out: common assembler and	a.out(4)
maintainer.	ar: archive and library	ar(1)
format.	ar: common archive file	ar(4)
language. bc:	arbitrary-precision arithmetic	bc(1)
acos: Fortran	arccosine intrinsic function.	acos(3F)
for portable archives. ar:	archive and library maintainer	ar(1)
cpio: format of cpio	archive.	cpio(4)
ar: common	archive file format.	ar(4)
archive:	archive files to tape.	archive(1)
ldahread: read the	archive header.	ldahread(3X)
tar: tape file	archiver.	tar(1)
maintainer for portable	archives. /archive and library	ar(1)
cpio: copy file	archives in and out.	cpio(1)
asin: Fortran	arcsine intrinsic function.	asin(3F)
atan2: Fortran	arctangent intrinsic function.	atan2(3F)
atan: Fortran	arctangent intrinsic function.	atan(3F)
imaginary part of complex	argument. aimag: Fortran	aimag(3F)
return Fortran command-line	argument. getarg:	getarg(3F)
varargs: handle variable	argument list.	varargs(5)
formatted output of a varags	argument list. vprintf: print	vprint(3S)
formatted output of a varargs	argument list. vprintf: print	vprintf(3X)
command.xargs: construct	argument list(s) and execute	xargs(1)
getopt: get option letter from	argument vector.	getopt(3C)
expr: evaluate	arguments as an expression.	expr(1)
echo: echo	arguments.	echo(1)
bc: arbitrary-precision	arithmetic language.	bc(1)
number facts.	arithmetic: provide drill in	arithmetic(6)
expr: evaluate arguments	as an expression.	expr(1)
	as: common assembler.	as(1)
characters. asa: interpret	ASA carriage control	asa(1)
control characters.	asa: interpret ASA carriage	asa(1)
ascii: map of	ASCII character set.	ascii(5)
set.	ascii: map of ASCII character	ascii(5)
long integer and base-64	ASCII string. /convert between	a641(3C)
intrinsic function.	asin: Fortran arcsine	asin(3F)
help:	ask for help.	help(1)
output. a.out: common	assembler and link editor	a.out(4)
as: common	assembler.	as(1)
assertion.	assert: verify program	assert(3X)
assert: verify program	assertion.	assert(3X)

setbuf:	assign buffering to a stream.	setbuf(3S)
intrinsic function.	atan: Fortran arctangent	atan(3F)
intrinsic function.	atan2: Fortran arctangent	atan2(3F)
interface. acu:	Automatic Call Unit (ACU)	acu(7)
wait:	await completion of process.	wait(1)
processing language.	awk: pattern scanning and	awk(1)
ungetc: push character	back into input stream.	ungetc(3S)
archive:	backup to streaming tape.	archive(1)
UNIX system file system	backup. /daily/weekly	filesave(1M)
finc: fast incremental	backup.	finc(1M)
frec: recover files from a	backup tape.	frec(1M)
print, initialize, update	bad information bdblk:	bdblk(1M)
	banner: make posters.	banner(1)
terminal capability data	base. terminfo:	terminfo(4)
between long integer and	base-64 ASCII string. /convert	a641(3C)
portions of path names.	basename, dirname: deliver	basename(1)
arithmetic language.	bd: arbitrary-precision	bc(1)
	bcopy: interactive block copy.	bcopy(1M)
update bad information	bdblk: print, initialize,	bdblk(1M)
	bdiff: big diff.	bdiff(1)
cb: C program	beautifier.	cb(1)
j0, j1, jn, y0, y1, yn:	Bessel functions.	bessel(3M)
	bfs: big file scanner.	bfs(1)
cpset: install object files in	binary directories.	cpset(1M)
fread:	binary input/output.	fread(3S)
bsearch:	binary search a sorted table.	bsearch(3C)
tsearch: manage	binary search trees.	tsearch(3C)
remove symbols and relocation	bits. strip:	strip(1)
bcopy: interactive	block copy.	bcopy(1M)
sum: print checksum and	block count of a file.	sum(1)
sync: update the super	block.	sync(1)
df: report number of free disk	blocks.	df(1M)
and, or xor, not: Fortran	boolean functions.	bool(3F)
UNIX system startup and	boot procedures. boot:	boot(8)
shell scripts.	brc: system initialization	brc(1M)
allocation.	brk: change data segment space	brk(2)
modest-sized programs.	bs: a compiler/interpreter for	bs(1)
sorted table.	bsearch: binary search a	bsearch(3C)
studio: standard	buffered input/output package.	stdio(3S)
setbuf: assign	buffering to a stream.	setbuf(3S)
mknod:	build special file.	mknod(1M)
swab: swap	bytes.	swab(3C)
cc:	C compiler.	cc(1)
cflow: generate	C flow graph.	cflow(1)
ccp: the	C language preprocessor.	cpp(1)
cb:	C program beautifier.	cb(1)
lint: a	C program checker.	lint(1)
cxref: generate	C program cross-reference.	cxref(1)
ctrace:	C program debugger.	ctrace(1)
	cal: print calendar.	cal(1)
dc: desk	calculator.	dc(1)
cal: print	calendar.	cal(1)
	calendar: reminder service.	calendar(1)
cu:	call another UNIX system.	cu(1C)
data returned by stat system	call. stat	stat(5)
acu: Automatic	Call Unit (ACU) interface.	acu(7)
intro: introduction to system	calls and error numbers.	intro(2)
link and unlink system	calls. link: exercise	link(1M)
to an LP line printer. lp,	cancel: send/cancel requests	lp(1)
terminfo: terminal	capability data base.	terminfo(4)

pnch: file format for	card images.	pnch(4)
asa: interpret ASA	carriage control characters.	asa(1)
files.	cat: concatenate and print	cat(1)
	cb: C program beautifier.	cb(1)
	cc: C compiler.	cc(1)
	cd: change working directory.	cd(1)
commentary of an SCCS delta.	cdc: change the delta	cdc(1)
value/ floor: floor,	ceiling, remainder, absolute	floor(3M)
	cflow: generate C flow graph.	cflow(1)
delta: make a delta	(change) to an SCCS file.	delta(1)
pipe: create an interprocess	channel.	pipe(2)
stream. ungetc: push	character back into input	ungetc(3S)
user. cuserid: get	character login name of the	cuserid(3S)
stream. getc: get	character or word from a	getc(3S)
putc: put	character or word on a stream.	putc(3S)
ascii: map of ASCII	character set.	ascii(5)
interpret ASA carriage control	characters. asa:	asa(1)
toupper: translate	characters.	conv(3C)
isalpha: classify	characters.	ctype(3)
tr: translate	characters.	tr(1)
for accounting.	chargefee: shell procedures	acctsh(1M)
directory.	chdir: change working	chdir(2)
fsck: file system consistency	check	fsck(1M)
checking procedure.	checkall: faster file system	checkall(1M)
lint: a C program	checker.	lint(1)
pwck: password/group file	checkers.	pwck(1M)
checkall: faster file system	checking procedure.	checkall(1M)
copy file systems with label	checking. volcopy:	volcopy(1M)
systems processed by fsck.	checklist: list of file	checklist(4)
file. sum: print	checksum and block count of a	sum(1)
times: get process and	child process times.	times(2)
terminate. wait: wait for	child process to stop or	wait(2)
	chmod: change mode.	chmod(1)
	chmod: change mode of file.	chmod(2)
of a file.	chown: change owner and group.	chown(2)
	chown: change owner or group.	chown(1)
	chroot: change root directory.	chroot(2)
for a command.	chroot: change root directory	chroot(1M)
isalpha:	classify characters.	ctype(3C)
uuclean: uucp spool directory	clean-up.	uuclean(1M)
clri:	clear i-node.	clri(1M)
alarm: set a process alarm	clock.	alarm(2)
cron:	clock daemon.	cron(1M)
	clock: report CPU time used.	clock(3C)
ldclose:	closes a common object file.	ldclose(3X)
close:	close a file descriptor.	close(2)
descriptor.	close: close a file	close(2)
fclose:	close or flush a stream.	fclose(3S)
	clri: clear i-node.	clri(1M)
	cmp: compare two files.	cmp(1)
line-feeds.	col: filter reverse	col(1)
	comb: combine SCCS deltas.	comb(1)
comb:	combine SCCS deltas.	comb(1)
common to two sorted files.	comm: select or reject lines	comm(1)
nice: run a	command at low priority.	nice(1)
change root directory for a	command. chroot:	chroot(1M)
env: set environment for	command execution.	env(1)
uux: UNIX-to-UNIX system	command execution.	(uux(1C)
system: issue a shell	command from Fortran.	system(3F)
quits. nohup: run a	command immune to hangups and	nohup(1)

ldlseek: seek to line number	entries of a section of a/	ldlseek(3X)
ldrseek: seek to relocation	entries of a section of a/	ldrseek(3X)
utmp: utmp and wtmp	entry formats.	utmp(4)
getgrent: get group file	entry.	getgrent(3C)
getpwent: get password file	entry.	getpwent(3C)
getutent: access utmp file	entry.	getut(3C)
name for file symbol table	entry. /retrieve symbol	ldgetname(3X)
the index of a symbol table	entry of a file. /compute	ldtbindex(3X)
read an indexed symbol table	entry of a file. ldtbread:	ldtbread(3X)
putpwent: write password file	entry.	putpwent(3C)
unlink: remove directory	entry.	unlink(2)
command execution.	env: set environment for	env(1)
	environ: user environment.	environ(5)
profile: setting up an	environment at login time.	profile(4)
environ: user	environment.	environ(5)
execution. env: set	environment for command	env(1)
getenv: return value for	environment name.	getenv(3C)
putenv: change or add value to	environment.	putenv(3C))
get: return Fortran	environment variable.	getenv(3F)
complementary error function.	erf: error function and	erf(3M)
	err: error-logging interface. . .	err(7)
from dump.	errdead: extract error records . .	errdead(1M)
daemon.	errdemon: error-logging	errdemon(1M)
format.	errfile: error-log file	errfile(4)
debugging. serverdaemon:	error analysis and network	serverdaemon(1
complementary error/ erf:	error function and	erf(3M)
function and complementary	error function. erf: error	erf(3M)
perror: system	error messages.	perror(3C)
to system calls and	error numbers. /introduction . . .	intro(2)
errdead: extract	error records from dump.	errdead(1M)
matherr:	error-handling function.	matherr(3M)
errfile:	error-log file format.	errfile(4)
errdemon:	error-logging daemon.	errdemon(1M)
errstop: terminate the	error-logging daemon.	errstop(1M)
err:	error-logging interface.	err(7)
process a report of logged	errors. errpt:	errpt(1M)
spell: find spelling	errors.	spell(1)
logged errors.	errpt: process a report of	errpt(1M)
error-logging daemon.	errstop: terminate the	errstop(1M)
terminal line/ dial:	establish an out-going	dial(3C)
setmnt:	establish mount table	setmnt(1M)
hypot:	Euclidean distance function. . . .	hypot(3M)
expression. expr:	evaluate arguments as an	expr(1)
test: condition	evaluation command.	test(1)
trace:	event-tracing driver.	trace(7)
	ex: text editor.	ex(1)
crash:	examine system images.	crash(1M)
	execl: execute a file.	exec(2)
network. net:	execute a command on the PCL . . .	net(1C)
execl:	execute a file.	exec(2)
netlet:	execute command on remote system .	netlet(1)
construct argument list(s) and	execute command. xargs:	xargs(1)
time. at:	execute commands at a later	at(1)
regcmp: compile and	execute regular expression. . . .	regcmp(3X)
set environment for command	execution. env:	env(1)
sleep: suspend	execution for an interval.	sleep(1)
sleep: suspend	execution for interval.	sleep(3C)
monitor: prepare	execution profile.	monitor(3C)
profil:	execution time profile.	profil(2)
UNIX-to-UNIX system command	execution. uux:	uux(1C)

system calls. link:	exercise link and unlink	link(1M)
a new file or rewrite an	existing one. creat: create	creat(2)
	exit: terminate process.	exit(2)
power, square root functions.	exp: exponential, logarithm, . . .	exp(3M)
intrinsic function.	exp: Fortran exponential	exp(3F)
pack: compress and	expand files.	pack(1)
conversion. int:	explicit Fortran type	ftype(3F)
function. exp: Fortran	exponential intrinsic	exp(3F)
square root functions. exp:	exponential, logarithm, power . . .	exp(3M)
expression.	expr: evaluate arguments as an . .	expr(1)
routines. regexp: regular	expression compile and match . . .	regexp(5)
regcmp: regular	expression compile.	regcmp(1)
expr: evaluate arguments as an	expression.	expr(1)
compile and execute regular	expression. regcmp:	regcmp(3X)
efl:	Extended Fortran Language.	efl(1)
dump. errdead:	extract error records from	errdead(1M)
	f77: Fortran 77 compiler.	f77(1)
factor:	factor a number.	factor(1)
	factor: factor a number.	factor(1)
data in a machine-independent	fashion /access long integer . . .	sputl(3X)
finc:	fast incremental backup.	finc(1M)
malloc:	fast main memory allocator. . . .	malloc(3X)
procedure. checkall:	faster file system checking . . .	checkall(1M)
abort: generate an IOT	fault.	abort(3C)
stream.	fclose: close or flush a	fclose(3S)
	fcntl: file control.	fcntl(2)
	fcntl: file control options. . . .	fcntl(5)
inquiries.	ferror: stream status	ferror(3S)
statistics for a file system.	ff: list file names and	ff(1M)
times. utime: set	file access and modification . . .	utime(2)
ldfcn: common object	file access routines.	ldfcn(4)
determine accessibility of a	file. access:	access(2)
tar: tape	file archiver.	tar(1)
cpio: copy	file archives in and out.	cpio(1)
pwck: password/group	file checkers.	pwck(1M)
chmod: change mode of	file.	chmod(2)
change owner and group of a	file. chown:	chown(2)
diff: differential	file comparator.	diff(1)
diff3: 3-way differential	file comparison.	diff3(1)
fcntl:	file control.	fcntl(2)
fcntl:	file control options.	fcntl(5)
public UNIX-to-UNIX system	file copy. uuto:	uuto(1C)
core: format of core image	file.	core(4)
umask: set and get	file creation mask.	umask(2)
crontab: user crontab	file.	crontab(1)
fields of each line of a	file. cut: cut out selected . . .	cut(1)
dd: convert and copy a	file.	dd(1)
a delta (change) to an SCCS	file. delta: make	delta(1)
close: close a	file descriptor.	close(2)
dup: duplicate an open	file descriptor.	dup(2)
	file: determine file type.	file(1)
selected parts of an object	file. dupm: dump	dump(1)
sact: print current SCCS	file editing activity.	sact(1)
getgrent: get group	file entry.	getgrent(3C)
getpwent: get password	file entry.	getpwent(3C)
getutent: access utmp	file entry.	getut(3C)
putpwent: write password	file entry.	putpwent(3C)
execl: execute a	file.	exec(2)
grep: search a	file for a pattern.	grep(1)
ldopen: open a common object	file for reading.	ldopen(3X)

acct: per-process accounting	file format.	acct(4)
ar: common archive	file format.	ar(4)
errfile: error-log	file format.	errfile(4)
pnch:	file format for card images.	pnch(4)
intro: introduction to	file formats.	intro(4)
line number entries of a	file function. /manipulate	ldlread(3X)
get: get a version of an SCCS	file.	get(1)
group: group	file.	group(4)
files. filehdr:	file header for common object	filehdr(4)
file. ldfhread: read the	file header of a common object	ldfhread(3X)
ldohseek: seek to the optional	file header of a file.	ldohseek(3X)
split: split a	file into pieces.	split(1)
issue: issue identification	file.	issue(4)
ldclose: close a common object	file.	ldclose(3X)
file header of a common object	file. ldfhread: read the	ldfhread(3X)
entries of a section of a	file. /seek to line number	ldlseek(3X)
the optional file header of a	file. ldohseek: seek to	ldohseek(3X)
entries of a section of a	file. /seek to relocation	ldrseek(3X)
section header of a	file. /read an indexed/named	ldshread(3X)
section of an object	file. /to an indexed/named	ldsseek(3X)
of a symbol table entry of a	file. /compute the index	ldtbindex(3X)
symbol table entry of a	file. /read an indexed	ldtbread(3X)
table of a common object	file. /seek to the symbol	ldtbseek(3X)
entries in a common object	file. linenum: line number	linenum(4)
lockf: record and	file locking	lockf(2)
link: link to a	file.	link(2)
mknod: build special	file.	mknod(1M)
or a special or ordinary	file. /make a directory,	mknod(2)
ctermid: generate	file name for terminal.	ctermid(3S)
mktemp: make a unique	file name.	mktemp(3C)
a file system. ff: list	file names and statistics for	ff(1M)
change the format of a text	file. newform:	newform(1)
name list of common object	file. nm: print	nm(1)
null: the null	file.	null(7)
/find the slot in the utmp	file of the current user.	ttyslot(3C)
/identify processes using a	file or file structure.	fuser(1M)
one. creat: create a new	file or rewrite an existing	creat(2)
passwd: password	file.	passwd(4)
merge same lines of several	file paste:	paste(1)
soft-copy terminals. pg:	file perusal filter for	pg(1)
fseek: reposition a	file pointer in a stream.	fseek(3S)
lseek: move read/write	file pointer.	lseek(2)
prs: print an SCCS	file.	prs(1)
read: read from	file.	read(2)
for a common object	file. /relocation information	reloc(4)
remove a delta from an SCCS	file. rmdel:	rmdel(1)
bfs: big	file scanner.	bfs(1)
two versions of an SCCS	file. sccsdiff: compare	sccsdiff(1)
sccsfile: format of SCCS	file.	sccsfile(4)
header for a common object	file. scnhdr: section	scnhdr(4)
stat: get	file status.	stat(2)
line number information from a	file. strip: strip symbol and	strip(1)
processes using a file or	file structure. /identify	fuser(1M)
checksum and block count of a	file. sum: print	sum(1)
/retrieve symbol name for	file symbol table entry.	ldgetname(3X)
syms: common object	file symbol table format.	syms(4)
daily/weekly UNIX system	file system backup. filesave:	filesave(1M)
procedure. checkall: faster	file system checking	checkall(1M)
fsck:	file system consistency check	fsck(1M)
fsdb:	file system debugger.	fsdb(1M)

names and statistics for a	file system. ff: list file	ff(1M)
volume.	file system: format of system	fs(4)
mkfs: construct a	file system.	mkfs(1M)
mount: mount and dismount	file system.	mount(1M)
mount: mount a	file system.	mount(2)
ustat: get	file system statistics.	ustat(2)
mnttab: mounted	file system table.	mnttab(4)
umount: unmount a	file system.	umount(2)
access time. dcopy: copy	file systems for optimal	dcopy(1M)
fsck. checklist: list of	file systems processed by	checklist(4)
checking. volcopy: copy	file systems with label	volcopy(1M)
deliver the last part of a	file. tail:	tail(1)
term: format of compiled term	file.	term(4)
tmpfile: create a temporary	file.	tmpfile(3S)
create a name for a temporary	file. tmpnam:	tmpnam(3S)
and modification times of a	file. touch: update access	touch(1)
ftw: walk a	file tree.	ftw(3C)
file: determine	file type.	file(1)
undo a previous get of an SCCS	file. unget:	unget(1)
report repeated lines in a	file. uniq:	uniq(1)
val: validate SCCS	file.	val(1)
write: write on a	file.	write(2)
umask: set	file-creation mode mask.	umask(1)
common object files.	filehdr: file header for	filehdr(4)
and print process accounting	file(s). acctcom: search	acctcom(1)
merge or add total accounting	files. acctmerg:	acctmerg(1M)
create and administer SCCS	files. admin:	admin(1)
cat: concatenate and print	files.	cat(1)
cmp: compare two	files.	cmp(1)
lines common to two sorted	files. comm: select or reject	comm(1)
cp, ln, mv: copy, link or move	files.	cp(1)
mark differences between	files. diffmk:	diffmk(1)
file header for common object	files. filehdr:	filehdr(4)
find: find	files.	find(1)
frec: recover	files from a backup tape.	frec(1M)
format specification in text	files. fspec:	fspec(4)
split f77, ratfor, or elf	files. fsplit:	fsplit(1)
string, format of graphical	files. /graphical primitive	gps(4)
cpset: install object	files in binary directories.	cpset(1M)
intro: introduction to special	files.	intro(7)
link editor for common object	files. ld:	ld(1)
rm: remove	files or directories.	rm(1)
pack: compress and expand	files.	pack(1)
pr: print	files.	pr(1)
section sizes of common object	files. size: print	size(1)
sort: sort and/or merge	files.	sort(1)
what: identify SCCS	files.	what(1)
system file system backup.	filesave: daily/weekly UNIX	filesave(1M)
terminals. pg: file perusal	filter for soft-copy	pg(1)
greek: select terminal	filter.	greek(1)
nl: line numbering	filter.	nl(1)
col:	filter reverse line-feeds.	col(1)
graphical device routines and	filters. hpd:	gdev(1G)
tplot: graphics	filters.	tplot(1G)
	finc: fast incremental backup.	finc(1M)
find:	find files.	find(1)
	find: find files.	find(1)
ttyname:	find name of a terminal.	ttyname(3C)
object library. lorder:	find ordering relation for an	lorder(1)
spell:	find spelling errors.	spell(1)

of the current user. ttyslot:	find the slot in the utmp file . .	ttyslot(3C)
tee: pipe	fitting.	tee(1)
string. ecvt: convert	floating-point number to	ecvt(3C)
frexp: manipulate parts of	floating-point numbers.	frexp(3C)
absolute value/ floor:	floor, ceiling, remainder	floor(3M)
remainder, absolute value/	floor: floor, ceiling, ...	floor(3M)
cflow: generate C	flow graph.	cflow(1)
fclose: close or	flush a stream.	fclose(3S)
	fopen: open a stream.	fopen(3S)
	fork: create a new process. . . .	fork(2)
per-process accounting file	format. acct:	acct(4)
RM05 disk packs. format:	format and/or check RPØ6 and . . .	format(1M)
ar: common archive file	format.	ar(4)
errfile: error-log file	format.	errfile(4)
pnch: file	format for card images.	pnch(4)
format:	format floppy disk drive	format(1M)
newform: change the	format of a text file.	newform(1)
inode:	format of an i-node.	inode(4)
term:	format of compiled term file. . .	term(4)
core:	format of core image file.	core(4)
cpio:	format of cpio archive.	cpio(4)
dir:	format of directories.	dir(4)
/graphical primitive string,	format of graphical files.	gps(4)
sccsfile:	format of SCCS file.	sccsfile(4)
file system:	format of system volume.	fs(4)
files. fspec:	format specification in text . . .	fspec(4)
object file symbol table	format. syms: common	syms(4)
intro: introduction to file	formats.	intro(4)
utmp: utmp and wtmp entry	formats.	utmp(4)
scanf: convert	formatted input.	scanf(3S)
argument list. vprintf: print	formatted output of a varargs . . .	vprintf(3S)
argument list. vprintf: print	formatted output of a varargs . . .	vprintf(3X)
printf: print	formatted output.	printf(3S)
f77:	Fortran 77 compiler.	f77(1)
abs:	Fortran absolute value.	abs(3F)
system/ signal: specify	Fortran action on receipt of a . .	signal(3F)
function. acos:	Fortran arccosine intrinsic	acos(3F)
function. asin:	Fortran arcsine intrinsic	asin(3F)
function. atan2:	Fortran arctangent intrinsic . . .	atan2(3F)
function. atan:	Fortran arctangent intrinsic . . .	atan(3F)
and, or, xor, not:	Fortran boolean functions.	bool(3F)
getarg: return	Fortran command-line argument. . .	getarg(3F)
intrinsic function. log1Ø:	Fortran common logarithm	log1Ø(3F)
intrinsic function. conjg:	Fortran complex conjugate	conjg(3F)
function. cos:	Fortran cosine intrinsic	cos(3F)
ratfor: rational	Fortran dialect.	ratfor(1)
getenv: return	Fortran environment variable. . .	getenv(3F)
function. exp:	Fortran exponential intrinsic . . .	exp(3F)
intrinsic function. cosh:	Fortran hyperbolic cosine	cosh(3F)
intrinsic function. sinh:	Fortran hyperbolic sine	sinh(3F)
intrinsic function. tanh:	Fortran hyperbolic tangent	tanh(3F)
complex argument. aimag:	Fortran imaginary part of	aimag(3F)
function. aint:	Fortran integer part intrinsic . .	aint(3F)
efl: Extended	Fortran Language.	efl(1)
functions. max:	Fortran maximum-value	max(3F)
functions. min:	Fortran minimum-value	min(3F)
intrinsic function. log:	Fortran natural logarithm	log(3F)
function. anint:	Fortran nearest integer	round(3F)
abort: terminate	Fortran program.	abort(3F)
functions. mod:	Fortran remaindering intrinsic . .	mod(3F)

ldahread: read the archive	header	ldahread(3X)
file. ldfhread: read the file	header of a common object	ldfhread(3X)
seek to the optional file	header of a file. ldohseek:	ldohseek(3X)
read an indexed/named section	header of a file. ldshread:	ldshread(3X)
	help: ask for help.	help(1)
help: ask for	help.	help(1)
handle special functions of	HP terminals. hp:	hp(1)
and filters.	hpd: graphical device routines	gdev(1G)
tables.	hsearch: manage hash search	hsearch(3C)
function. cosh: Fortran	hyperbolic cosine intrinsic	cosh(3F)
sinh:	hyperbolic functions.	sinh(3M)
function. sinh: Fortran	hyperbolic sine intrinsic	sinh(3F)
function. tanh: Fortran	hyperbolic tangent intrinsic	tanh(3F)
function.	hypot: Euclidean distance	hypot(3M)
	iargc.	iargc(3F)
disk accounting data by user	ID. diskusg: generate	diskusg(1M)
and names.	id: print user and group IDs	id(1)
setpgrp: set process group	ID.	setpgrp(2)
issue: issue	identification file.	issue(4)
file or file/ fuser:	identify processes using a	fuser(1M)
what:	identify SCCS files.	what(1)
id: print user and group	IDs and names.	id(1)
group. and parent process	IDs. /get process, process	getpid(2)
getuid: get user	IDs	getuid(2)
setgid: set user and group	IDs. setuid,	setuid(2)
core: format of core	image file.	core(4)
crash: examine system	images.	crash(1M)
pnch: file format for card	images.	pnch(4)
argument. aimag: Fortran	imaginary part of complex	aimag(3F)
nohup: run a command	immune to hangups and quits.	nohup(1)
finc: fast	incremental backup.	finc(1M)
of a/ ldtbindex: compute the	index of a symbol table entry	ldtbindex(3X)
Fortran substring	index: return location of	index(3F)
a file. ldtbread: read an	indexed symbol table entry of	ldtbread(3X)
of a file. ldshread: read an	indexed/named section header	ldshread(3X)
object/ ldsseek: seek to an	indexed/named section of an	ldsseek(3X)
initialization.	init: process control	init(1M)
inittab: script for the	init process.	inittab(4)
init: process control	initialization.	init(1M)
brc: system	initialization shell scripts.	brc(1M)
information bdblk: print,	initialize, update bad	bdblk(1M)
process. popen:	initiate pipe to/from a	popen(3S)
process.	inittab: script for the init	inittab(4)
clri: clear	i-node.	clri(1M)
	inode: format of an i-node.	inode(4)
inode: format of an	i-node.	inode(4)
scanf: convert formatted	input.	scanf(3S)
push character back into	input stream. ungetc:	ungetc(3S)
fread: binary	input/output.	fread(3S)
stdio: standard buffered	input/output package.	stdio(3S)
ferror: stream status	inquiries.	ferror(3S)
uustat: uucp status	inquiry and job control.	uustat(1C)
install:	install commands.	install(1M)
	install: install commands.	install(1M)
directories. cpset:	install object files in binary	cpset(1M)
conversion.	int: explicit Fortran type	ftype(3F)
abs: return	integer absolute value.	abs(3C)
a64l: convert between long	integer and base-64 ASCII/	a64l(3C)
sputl: access long	integer data in a/	sputl(3X)
anint: Fortran nearest	integer functions.	round(3F)

function. aint: Fortran	integer part intrinsic	aint(3F)
strtol: convert string to	integer.	strtol(3C)
l3tol: convert between 3-byte	integers and long integers.	l3tol(3C)
3-byte integers and long	integers. /convert between	l3tol(3C)
bcopy:	interactive block copy	bcopy(1M)
system. mailx:	interactive message processing	mailx(1)
acu: Automatic Call Unit (ACU)	interface.	acu(7)
err: error-logging	interface.	err(7)
plot: graphics	interface.	plot(4)
plot: graphics	interface subroutines.	plot(3X)
termio: general terminal	interface.	termio(7)
tty: controlling terminal	interface.	tty(7)
spline:	interpolate smooth curve.	spline(1G)
characters. asa:	interpret ASA carriage control	asa(1)
sno: SNOBOL	interpreter.	sno(1)
pipe: create an	interprocess channel.	pipe(2)
facilities/ ipcs: report	inter-process communication	ipcs(1)
package. ftok: standard	interprocess communication	stdipc(3C)
suspend execution for an	interval. sleep:	sleep(1)
sleep: suspend execution for	interval.	sleep(3C)
acos: Fortran arccosine	intrinsic function.	acos(3F)
aint: Fortran integer part	intrinsic function.	aint(3F)
asin: Fortran arcsine	intrinsic function.	asin(3F)
atan2: Fortran arctangent	intrinsic function.	atan2(3F)
atan: Fortran arctangent	intrinsic function.	atan(3F)
Fortran complex conjugate	intrinsic function. conjg:	conjg(3F)
cos: Fortran cosine	intrinsic function.	cos(3F)
Fortran hyperbolic cosine	intrinsic function. cosh:	cosh(3F)
double precision product	intrinsic function. drpod:	dprod(3F)
exp: Fortran exponential	intrinsic function.	exp(3F)
Fortran common logarithm	intrinsic function. log1Ø:	log1Ø(3F)
log: Fortran natural logarithm	intrinsic function.	log(3F)
sign: Fortran transfer-of-sign	intrinsic function.	sign(3F)
sin: Fortran sine	intrinsic function.	sin(3F)
sinh: Fortran hyperbolic sine	intrinsic function.	sinh(3F)
sqrt: Fortran square root	intrinsic function.	sqrt(3F)
tan: Fortran tangent	intrinsic function.	tan(3F)
Fortran hyperbolic tangent	intrinsic function. tanh:	tanh(3F)
dim: positive difference	intrinsic functions.	dim(3F)
mod: Fortran remaindering	intrinsic functions.	mod(3F)
lge: string comparison	intrinsic functions.	strcmp(3F)
commands	intro: introduction to	intro(1)
formats.	intro: introduction to	intro(4)
maintenance commands	intro: introduction to	intro(1M)
miscellany.	intro: introduction to	intro(5)
files.	intro: introduction to	intro(7)
subroutines and libraries.	intro: introduction to	intro(3)
calls and error numbers.	intro: introduction to system	intro(2)
maintenance procedures.	intro: introduction to system	intro(8)
intro:	introduction to commands	intro(1)
intro:	introduction to file formats.	intro(4)
commands intro:	introduction to maintenance	intro(1M)
intro:	introduction to miscellany.	intro(5)
intro:	introduction to special files	intro(7)
and libaries. intro:	introduction to subroutines	intro(3)
and error numbers. intro:	introduction to system calls	intro(2)
maintenance/ intro:	introduction to system	intro(8)
ncheck: generate names from	i-numbers.	ncheck(1M)
	ioctl: control device.	ioctl(2)
abort: generate an	IOT fault.	abort(3C)

	ipcrm: remove a message queue	ipcrm(1)
communication facilities/	ipcs: report inter-process	ipcs(1)
generator.	irand: random number	rand(3F)
	isalpha: classify characters	ctype(3C)
Fortran. system:	issue a shell command from	system(3F)
system:	issue a shell command.	system(3S)
issue:	issue identification file.	issue(4)
file.	issue: issue identification	issue(4)
news: print news	items.	news(1)
functions.	jØ, j1, jn, yØ, y1, yn: Bessel	bessel(3M)
functions. jØ,	j1, jn, yØ, y1, yn: Bessel	bessel(3M)
functions. jØ, j1,	jn, yØ, y1, yn: Bessel	bessel(3M)
operator.	join: relational database	join(1)
makekey: generate encryption	key.	makekey(1)
killall:	kill all active processes.	killall(1M)
process or a group of/	kill: send a signal to a	kill(2)
	kill: terminate a process.	kill(1)
processes.	killall: kill all active	killall(1M)
integers and long integers.	l3tol: convert between 3-byte	l3tol(3C)
copy file systems with	label checking. volcopy:	volcopy(1M)
scanning and processing	language. awk: pattern	awk(1)
arbitrary-precision arithmetic	language. bc:	bc(1)
efl: Extended Fortran	Language.	efl(1)
cpp: the C	language preprocessor.	cpp(1)
command programming	language. /standard/restricted	sh(1)
shl: shell	layer manager.	shl(1)
object files.	ld: link editor for common	ld(1)
header	ldahread: read the archive	ldahread(3X)
file.	ldclose: close a common object	ldclose(3X)
access routines.	ldfcn: common object file	ldfcn(4)
of a common object file.	ldfhread: read the file header	ldfhread(3X)
name for file symbol table/	ldgetname: retrieve symbol	ldgetname(3X)
number entries of a file/	ldlread: manipulate line	ldlread(3X)
entries of a section of a/	ldlseek: seek to line number	ldlseek(3X)
file header of a file	ldohseek: seek to the optional	ldohseek(3X)
file for reading.	ldopen: open a common object	ldopen(3X)
entries of a section of a/	ldrseek: seek to relocation	ldrseek(3X)
indexed/named section header/	ldshread: read an	ldshread(3X)
indexed/named section of an/	ldsseek: seek to an	ldsseek(3X)
of a symbol table entry of a/	ldtbindex: compute the index	ldtbindex(3X)
symbol table entry of a file.	ldtbread: read an indexed	ldtbread(3X)
table of a common object/	ldtbseek: seek to the symbol	ldtbseek(3X)
string.	len: return length of Fortran	len(3F)
len: return	length of Fortran string.	len(3F)
getopt: get option	letter from argument vector.	getopt(3C)
simple lexical tasks.	lex: generate programs for	lex(1)
generate programs for simple	lexical tasks. lex:	lex(1)
intrinsic functions.	lge: string comparision	strcmp(3F)
to subroutines and	libraries. /introduction	intro(3)
relation for an object	library. /find ordering	lorder(1)
portable/ ar: archive and	library maintainer for	ar(1)
ulimit: get and set user	limits.	ulimit(2)
an out-going-terminal	line connection. /establish	dial(3C)
type, modes, speed, and	line discipline. /set terminal	getty(1M)
line: read one	line.	line(1)
common object file. linenum:	line number entries in a	linenum(4)
function. ldlread: manipulate	line number entries of a file	ldlread(3X)
section of a/ ldlseek: seek to	line number enteries of a	ldlseek(3X)
file. strip: strip symbol and	line number information from a	strip(1)
nl:	line numbering filter.	nl(1)

out selected fields of each	line of a file. cut: cut	cut(1)
send/cancel requests to an LP	line printer. lp, cancel:	lp(1)
lp:	line printer.	lp(7)
	line: read one line.	line(1)
lsearch:	linear search and update.	lsearch(3C)
col: filter reverse	line-feeds.	col(1)
in a common object file.	linenum: line number entries	linenum(4)
files. comm: select or reject	lines common to two sorted	comm(1)
uniq: report repeated	lines in a file.	uniq(1)
paste: merge same	lines of several file	paste(1)
link: exercise	link and unlink system calls	link(1M)
files. ld:	link editor for common object	ld(1)
a.out: common assembler and	link editor output.	a.out(4)
system calls.	link: exercise link and unlink	link(1M)
	link: link to a file.	link(2)
cp, ln, mv: copy,	link or move files.	cp(1)
link:	link to a file.	link(2)
	lint: a C program checker.	lint(1)
ls:	list contents of directory.	ls(1)
for a file system. ff:	list file names and statistics	ff(1M)
nlist: get entries from name	list.	nlist(3C)
nm: print name	list of common object file.	nm(1)
by fsck. checklist:	list of file systems processed	checklist(4)
handle variable argument	list. varargs:	varargs(5)
output of a varargs argument	list. /print formatted	vprintf(3S)
output of a varargs argument	list. /print formatted	vprintf(3X)
xargs: construct argument	list(s) and execute command.	xargs(1)
files. cp,	ln, mv: copy, link or move	cp(1)
index: return	location of Fortran substring.	index(3F)
end: last	locations in program.	end(3C)
memory. plock:	lock process, text, or data in	plock(2)
intrinsic function.	log: Fortran natural logarithm	log(3F)
gamma:	log gamma function.	gamma (3M)
newgrp:	log in to a new group.	newgrp(1)
logarithm intrinsic function.	log1Ø: Fortran common	log1Ø(3F)
log1Ø: Fortran common	logarithm intrinsic function.	log1Ø(3F)
log: Fortran natural	logarithm intrinsic function.	log(3F)
functions. exp: exponential,	logarithm, power, square root	exp(3M)
errpt: process a report of	logged errors.	errpt(1M)
getlogin: get	login name.	getlogin(3C)
logname: get	login name.	logname(1)
cuserid: get character	login name of the user.	cuserid(3S)
logname: return	login name of user.	logname(3X)
passwd: change	login password.	passwd(1)
	login: sign on.	login(1)
setting up an environment at	login time. profile:	profile(4)
	logname: get login name.	logname(1)
user.	logname: return login name of	logname(3X)
string. a641: convert between	long integer and base-64 ASCII	a641(3X)
sputl: access	long integer data in a/	sputl(3X)
between 3-byte integers and	long integers. l3tol: convert	l3tol(3C)
for an object library.	lorder: find ordering relation	lorder(1)
nice: run a command at	low priority.	nice(1)
requests to an LP line/	lp, cancel: send/cancel	lp(1)
send/cancel requests to an	LP line printer. lp, cancel:	lp(1)
	lp: line printer.	lp(7)
enable: enable/disable	LP printers.	enable(1)
lpsched: start/stop the	LP request scheduler	lpsched(1M)
accept: allow/prevent	LP requests.	accept(1M)
lpadmin: configure the	LP spooling system.	lpadmin(1M)

lpstat: print	LP status information.	lpstat(1)
spooling system.	lpadmin: configure the LP	lpadmin(1M)
request scheduler	lpsched: start/stop the LP	lpsched(1M)
information.	lpstat: print LP status	lpstat(1)
directory.	ls: list contents of	ls(1)
update.	lsearch: linear search and	lsearch(3C)
pointer.	lseek: move read/write file	lseek(2)
	m4: macro processor.	m4(1)
values:	machine-dependent values.	values(5)
/access long integer data in a	machine-independent fashion	sputl(3X)
m4:	macro processor.	m4(1)
send mail to user or read	mail. mail:	mail(1)
read mail.	mail: send mail to users or	mail(1)
mail: send	mail to users or read mail. . . .	mail(1)
processing system.	mailx: interactive message	mailx(1)
malloc:	main memory allocator.	malloc(3C)
malloc: fast	main memory allocator.	malloc(3X)
regenerate groups of/ make:	maintain, update, and	make(1)
ar: archive and library	maintainer for portable/	ar(1)
intro: introduction to	maintenance commands	intro(1M)
intro: introduction to system	maintenance procedures.	intro(8)
SCCS file. delta:	make a delta (change) to an	delta(1)
mkdir:	make a directory.	mkdir(1)
or ordinary file. mknod:	make a directory, or a special . .	mknod(2)
mktemp:	make a unique file name.	mktemp(3C)
regenerate groups of/	make: maintain, update, and	make(1)
banner:	make posters.	banner(1)
key.	makekey: generate encryption . . .	makekey(1)
allocator.	malloc: fast main memory	malloc(3X)
	malloc: main memory allocator. . .	malloc(3C)
manual.	man: print entries in this	man(1)
tsearch:	manage binary search trees. . . .	tsearch(3C)
hsearch:	manage hash search tables.	hsearch(3C)
shl: shell layer	manager.	shl(1)
records. fwtmp:	manipulate connect accounting . . .	fwtmp(1M)
of a file function. ldlread:	manipulate line number entries . .	ldlread(3X)
floating-point/ frexp	manipulate parts of	frexp(3C)
man: print entries in this	manual.	man(1)
ascii:	map of ASCII character set. . . .	ascii(5)
files. diffmk:	mark differences between	diffmk(1)
umask: set file-creation mode	mask.	umask(1)
set and get file creation	mask. umask:	umask(2)
table. master:	master device information	master(4)
information table.	master: master device	master(4)
regular expression compile and	match routines. regexp:	regexp(5)
math:	math functions and constants. . .	math(5)
constants.	math: math functions and	math(5)
function.	matherr: error-handling	matherr(3M)
multiple-access-user-space/	maus:	maus(2)
functions.	max: Fortran maximum-value	max(3F)
max: Fortran	maximum-value functions.	max(3F)
accounting.	mclock: return Fortran time	mclock(3F)
	mem: core memory.	mem(7)
	memccpy: memory operations. . . .	memory(3C)
malloc: main	memory allocator.	malloc(3C)
malloc: fast main	memory allocator.	malloc(3X)
shmctl: shared	memory control operations.	shmctl(2)
mem: core	memory.	mem(7)
/(shared	memory) operations.	maus(2)
memccpy:	memory operations.	memory(3C)

acc disk:	overview of accounting	acct(1M)
chown: change	owner and group of a file.	chown(2)
chown: change	owner or group.	chown(1)
files.	pack: compress and expand	pack(1)
handling and optimization	package. curses: CRT screen	curses(3X)
sal: system activity report	package.	sar(1M)
standard buffered input/output	package. stdio:	stdio(3S)
interprocess communication	package. ftok: standard	stdipc(3C)
4014 terminal. 4014:	paginator for the TEKTRONIX	4014(1)
process, process group, and	parent process IDs. /get	getpid(2)
getopt:	parse command options.	getopt(1)
	passwd: change login password.	passwd(1)
	passwd: password file.	passwd(4)
getpwent: get	password file entry.	getpwent(3C)
putpwent: write	password file entry.	putpwent(3C)
passwd:	password file.	passwd(4)
getpass: read a	password.	getpass(3C)
passwd: change login	password.	passwd(1)
pwck:	password/group file checker.	pwck(1M)
several file	paste: merge same lines of	paste(1)
dirname: deliver portions of	path names. basename,	basename(1)
directory. getcwd: get	path-name of current working	getcwd(3C)
grep: search a file for a	pattern.	grep(1)
processing language. awk:	pattern scanning and	awk(1)
signal.	pause: suspend process until	pause(2)
mesg:	permit or deny messages.	mesg(1)
format. acct:	per-process accounting file	acct(4)
acctcms: command summary from	per-process accounting/	acctcms(1M)
	perror: system error messages.	perror(3C)
terminals. pg: file	perusal filter for soft-copy	pg(1)
soft-copy terminals.	pg: file perusal filter for	pg(1)
split: split a file into	pieces.	split(1)
channel.	pipe: create an interprocess	pipe(2)
tee:	pipe fitting.	tee(1)
popen: initiate	pipe to/from a process.	popen(3S)
data in memory.	plock: lock process, text, or	plock(2)
	plot: graphics interface.	plot(4)
subroutines	plot: graphics interface.	plot(3X)
images.	pnch: file format for card	pnch(4)
fseek: reposition a file	pointer in a stream.	fseek(3S)
lseek: move read/write file	pointer.	lseek(2)
process.	popen: initiate pipe to/from a	popen(3S)
and library maintainer for	portable archives. /archive	ar(1)
basename, dirname: deliver	portions of path names.	basename(2)
functions. dim:	positive difference intrinsic	dim(3F)
banner: make	posters.	banner(1)
exp: exponential, logarithm,	power, square root functions.	exp(3M)
	pr: print files.	pr(1)
functions. dprod: double	precision product intrinsic	dprod(3F)
monitor:	prepare execution profile.	monitor(3C)
cpp: the C language	processor.	cpp(1)
unget: undo a	previous get of an SCCS file.	unget(1)
profiler.	prf: operating system	prf(7)
profiler.	prfld: operating system	profiler(1M)
graphical/ gps: graphical	primitive string, format of	gps(4)
types:	primitive system data types.	types(5)
prs:	print an SCCs file.	prs(1)
date:	print and set the date.	date(1)
cal:	print calendar.	cal(1)
of a file. sum:	print checksum and block count	sum(1)

editing activity. sact:	print current SCCS file	sact(1)
man:	print entries in this manual.	man(1)
cat: concatenate and	print files.	cat(1)
pr:	print files.	pr(1)
varargs argument/ vprintf:	print formatted output of a	vprintf(3S)
varargs argument/ vprintf:	print formatted output of a	vprintf(3X)
printf:	print formatted output	printf(3S)
information bdblk:	print, initialize, update bad	bdblk(1M)
lpstat:	print LP status information.	lpstat(1)
object file. nm:	print name list of common	nm(1)
system. uname:	print name of current UNIX	uname(1)
news:	print news items.	news(1)
file(s). acctcom: search and	print process accounting	acctcom(1)
object files. size:	print section sizes of common	size(1)
names. id:	print user and group IDs and	id(1)
requests to an LP line	printer. /cancel: send/cancel	lp(1)
lp: line	printer.	lp(7)
enable: enable/disable LP	printers.	enable(1)
output.	printf: print formatted	printf(3S)
nice: run a command at low	priority.	nice(1)
nice: change	priority of a process	nice(2)
errors. errpt:	process a report of logged	errpt(1M)
acct: enable or disable	process accounting.	acct(2)
acctprc1:	process accounting.	acctprc(1M)
acctcom: search and print	process accounting file(s).	acctcom(1)
alarm: set a	process alarm clock.	alarm(2)
times. times: get	process and child process	times(2)
initialization. init:	process control	init(1M)
timex: time a command; report	process data and system/	timex(1)
exit: terminate	process.	exit(2)
fork: create a new	process.	fork(2)
process/ getpid: get process	process group, and parent	getpid(2)
setpgrp: set	process group ID.	setpgrp(2)
process group, and parent	process IDs. /get process,	getpid(2)
inittab: script for the init	process.	inittab(4)
kill: terminate a	process.	kill(1)
nice: change priority or a	process.	nice(2)
kill: send a signal to a	process or a group of/	kill(2)
popen: initiate pipe to/from a	process.	popen(3S)
parent process/ getpid: get	process, process group, and	getpid(2)
ps: report	process status.	ps(1)
memory. plock: lock	process, text, or data in	plock(2)
times: get process and child	process times.	times(2)
wait: wait for child	process to stop or terminate.	wait(2)
ptrace:	process trace.	ptrace(2)
pause: suspend	process until signal.	pause(2)
wait: await completion of	process.	wait(1)
list of file systems	processed by fsck. checklist:	checklist(4)
to a process or a group of	processes. /send a signal	kill(2)
killall: kill all active	processes.	killall(1M)
structure. fuser: identify	processes using a file or file	fuser(1M)
awk: pattern scanning and	processing language.	awk(1)
shutdown: terminate all	processing.	shutdown(1M)
mailx: interactive message	processing system.	mailx(1)
m4: macro	processor.	m4(1)
provide truth value about	processor type.	machid(1)
dprod: double precision	product intrinsic function.	dprod(3F)
	prof: display profile data.	prof(1)
function.	prof: profile within a	prof(5)
profile.	profil: execution time	profil(2)

prof: display	profile data.	prof(1)
monitor: prepare execution	profile.	monitor(3C)
profil: execution time	profile.	profil(2)
environment at login time	profile: setting up an	profile(4)
prof:	profile within a function.	prof(5)
prf: operating system	profiler.	prf(7)
prfld: operating system	profiler.	profiler(1M)
sadp: disk access	profiler.	sadp(1M)
standard/restricted command	programming language. /the	sh(1)
processor type:	provide truth value about	machid(1)
true:	provide truth values.	true(1)
	prs: print an SCCS file	prs(1)
	ps: report process status.	ps(1)
sxt:	pseudo-device driver.	sxt(7)
/generate uniformly distributed	pseudo-random numbers.	drand48(3C)
	ptrace: process trace.	ptrace(2)
stream. ungetc:	push character back into input . .	ungetc(3S)
a stream.	putc: put character or word on . .	putc(3S)
environment.	putenv: change or add value to . .	putenv(3C)
entry.	putpwent: write password file . . .	putpwent(3C)
stream.	puts: put a string on a	puts(3S)
checkers.	pwck: password/group file	pwck(1M)
	pwd: working directory name. . . .	pwd(1)
	qsort: quicker sort.	qsort(3C)
tput:	query terminfo database.	tput(1)
ipcrm: remove a message	queue	ipcrm(1)
msgget: get message	queue	msgget(2)
qsort:	quicker sort.	qsort(3C)
command immune to hangups and	quits. nohup: run a	nohup(1)
generator.	rand: simple random-number	rand(3C)
irand:	random number generator.	rand(3F)
rand: simple	random-number generator.	rand(3C)
fsplit: split f77,	ratfor, or efl files.	fsplit(1)
dialect.	ratfor: rational Fortran	ratfor(1)
ratfor:	rational Fortran dialect.	ratfor(1)
getpass:	read a password.	getpass(3C)
entry of a file. ldtbread:	read an indexed symbol table . . .	ldtbread(3X)
header of a file. ldshread:	read an indexed/named section . . .	ldshread(3X)
read:	read from file.	read(2)
mail: send mail to users or	read mail.	mail(1)
line:	read one line.	line(1)
	read: read from file.	read(2)
ldahread:	read the archive header	ldahread(3X)
common object file. ldfhread:	read the file header of a	ldfhread(3X)
open a common object file for	reading. ldopen:	ldodpen(3X)
open: open for	reading or writing.	open(2)
lseek: move	read/write file pointer.	lseek(2)
specify what to do upon	receipt of a signal. signal: . . .	signal(2)
/specify Fortran action on	receipt of a system signal. . . .	signal(3F)
lockf:	record and file locking.	lockf(2)
from per-process accounting	records. /command summary	acctcms(1M)
errdead: extract error	records from dump.	errdead(1M)
manipulate connect accounting	records. fwtmp:	fwtmp(1M)
tape. frec:	recover files from a backup	frec(1M)
ed,	red: text editor.	ed(1)
regular expression	regcmp: compile and execute	regcmp(3X)
compile.	regcmp: regular expressions	regcmp(1)
make: maintain, update, and	regenerate groups of programs. . .	make(1)
compile and match routines.	regexp: regular expression	regexp(5)
match routines. regexp:	regular expression compile and . .	regexp(5)

regcmp:	regular expression compile. . . .	regcmp(1)
regcmp: compile and execute	regular expression.	regcmp(3X)
sorted files. comm: select or	reject lines common to two	comm(1)
lorder: find ordering	relation for an object/	lorder(1)
join:	relational database operator. . .	join(1)
for a common object file.	reloc: relocation information . . .	reloc(4)
strip: remove symbols and	relocation bits.	strip(1)
section of a/ ldrseek: seek to	relocation entries of a	ldrseek(3X)
common object file. reloc:	relocation information for a . . .	reloc(4)
floor: floor. ceiling,	remainder, absolute value/	floor(3M)
functions. mod: Fortran	remaindering intrinsic	mod(3F)
calendar:	reminder service.	calendar(1)
netlet: execute command on	remote system.	netlet(1)
run: run command on a	remote system.	run(1)
ct: spawn getty to a	remote terminal.	ct(1C)
file. rmdel:	remove a delta from an SCCS	rmdel(1)
ipcrm:	remove a message gueue	ipcrm(1)
unlink:	remove directory entry.	unlink(2)
rm:	remove files or directores. . . .	rm(1)
bits. strip:	remove symbols and relocation . . .	strip(1)
uniq: report	repeated lines in a file.	uniq(1)
clock:	report CPU time used.	clock(3C)
communication/ ipcs:	report inter-process	ipcs(1)
blocks. df:	report number of free disk	df(1M)
errpt: process a	report of logged errors.	errpt(1M)
sal: system activity	report package.	sar(1M)
timex: time a command:	report process data and system/ . .	timex(1)
ps:	report process status.	ps(1)
file. uniq:	report repeated lines in a	uniq(1)
trenter: enter a trouble	report.	trenter(1)
sar: system activity	reporter.	sar(1)
stream. fseek:	reposition a file pointer in a . .	fseek(3S)
lpsched: start/stop the LP	request scheduler	lpsched(1M)
accept: allow/prevent LP	requests.	accept(1M)
lp, cancel: send/cancel	requests to an LP line/	lp(1)
recover:	restore files from tape to disk . .	recover(1)
symbol table/ ldgetname:	retrieve symbol name for file . . .	ldgetname(3X)
argument. getarg:	return Fortran command-line	getarg(3F)
variable. getenv:	return Fortran environment	getenv(3F)
accounting. mclock:	return Fortran time	mclock(3F)
abs:	return integer absolute value. . .	abs(3C)
string. len:	return length of Fortran	len(3F)
substring. index:	return location of Fortran	index(3F)
logname:	return login name of user.	logname(3X)
name. getenv:	return value for environment . . .	getenv(3C)
stat: data	returned by stat system call. . .	stat(5)
col: filter	reverse line-feeds.	col(1)
creat: create a new file or	rewrite an existing one.	creat(2)
directories.	rm: remove files or	rm(1)
SCCS file	rmdel: remove a delta from an . . .	rmdel(1)
chroot: change	root directory.	chroot(2)
chroot: change	root directory for a command. . .	chroot(1M)
logarithm. power, square	root functions. /exponential, . . .	exp(3M)
sqrt: Fortran square	root intrinsic function.	sqrt(3F)
hpd: graphical device	routines and filters.	gdev(1G)
common object file access	routines. ldfcn:	ldfcn(4)
expression compile and match	routines. regexp: regular	regexp(5)
graphical table of contents	routines. toc:	toc(1G)
nice:	run a command at low priority. . .	nice(1)
hangups and quits. nohup:	run a command immune to	nohup(1)

run:	run command on remote system. . .	run(1)
runacct:	run daily accounting.	runacct(1M)
	runacct: run daily accounting. . .	runacct(1M)
package.	sal: system activity report	sar(1M)
editing activity.	sact: print current SCCS file . . .	sact(1)
	sadp: disk access profiler. . . .	sadp(1M)
	sag: system activity graph. . . .	sag(1G)
	sar: system activity reporter. . .	sar(1)
input.	scanf: convert formatted	scanf(3S)
bfs: big file	scanner.	bfs(1)
language. awk: pattern	scanning and processing	swk(1)
the delta commentary of an	SCCS delta. cdc: change	cdc(1)
comb: combine	SCCS deltas.	comb(1)
make a delta (change) to an	SCCS file. delta:	delta(1)
sact: print current	SCCS file editing activity. . . .	sact(1)
get: get a version of an	SCCS file.	get(1)
prs: print an	SCCS file.	prs(1)
rmdel: remove a delta from an	SCCS file.	rmdel(1)
compare two versions of an	SCCS file. sccsdiff:	sccsdiff(1)
sccsfile: format of	SCCS file.	sccsfile(4)
undo a previous get of an	SCCS file. unget:	unget(1)
val: validate	SCCS file.	val(1)
admin: create and administer	SCCS files.	admin(1)
what: identify	SCCS files.	what(1)
of an SCCS file	sccsdiff: compare two versions . .	sccsdiff(1)
	sccsfile: format of SCCS file. . .	sccsfile(4)
start/stop the LP request	scheduler lpsched:	lpsched(1M)
common object file.	scnhdr: section header for a . . .	scnhdr(4)
optimizatioin/ curses: CRT	screen handling and	curses(3X)
vi:	screen-oriented editor	vi(1)
inittab:	script for the init process. . . .	inittab(4)
system initialization shell	scripts. brc:	brc(1M)
	sdb: symbolic debugger.	sdb(1)
program.	sdiff: side-by-side difference . .	sdiff(1)
grep:	search a file for a pattern. . . .	grep(1)
bsearch: binary	search a sorted table.	bsearch(3C)
accounting file(s). acctcom:	search and print process	acctcom(1)
lsearch: linear	search and update.	lsearch(3C)
hsearch: manage hash	search tables.	hsearch(3C)
tsearch: manage binary	search trees.	tsearch(3C)
object file. scnhdr:	section header for a common	scnhdr(4)
/read an indexed/named	section header of a file.	ldshread(3X)
to line number entries of a	section of a file. /seek	ldlseek(3X)
to relocation entries of a	section of a file. /seek	ldrseek(3X)
/seek to an indexed/named	section of an object file.	ldsseek(3X)
files. size: print	section sizes of common object . .	size(1)
	sed: stream editor.	sed(1)
section of an object/ ldsseek:	seek to an indexed/named	ldsseek(3X)
a section of a file. lslseek:	seek to line number entries of . .	ldlseek(3X)
a section of a file. ldrseek:	seek to relocation entries of . . .	ldrseek(3X)
header of a file. ldohseek:	seek to the optional file	ldohseek(3X)
common object file. ldtbseek:	seek to the symbol table of a . . .	ldtbseek(3X)
shmget: get shared memory	segment.	shmget(2)
brk: change data	segment space allocation.	brk(2)
to two sorted files. comm:	select or reject lines common . .	comm(1)
greek:	select terminal filter.	greek(1)
of a file. cut: cut out	selected fields of each line . . .	cut(1)
file. dump: dump	selected parts of an object	dump(1)
semctl:	semaphore control operations. . .	semctl(2)
semop:	semaphore operations.	semop(2)

semget: get set of	semaphores.	semget(2)
operations.	semctl: semaphore control	semctl(2)
semget:	get set of semaphores.	semget(2)
	semop: semaphore operations.	semop(2)
a group of processes. kill:	send a signal to a process or	kill(2)
mail. mail:	send mail to users or read	mail(1)
line printer. lp, cancel:	send/cancel requests to an LP	lp(1)
stream.	setbuf: assign buffering to a	setbuf(3S)
IDs. setuid,	setgid: set user and group	setuid(2)
	setjmp: non-local goto.	setjmp(3C)
	setmnt: establish mount table.	setmnt(1M)
	setpgrp: set process group ID.	setpgrp(2)
login time. profile:	setting up an environment at	profile(4)
gettydef: speed and terminal	settings used by getty.	gettydefs(4)
group IDs.	setuid, setgid: set user and	setuid(2)
standard/restricted command/	sh: shell, the	sh(1)
operations. shmctl:	shared memory control	shmctl(2)
/multiple-access-user-space	(shared memory) operations.	maus(2)
shmop:	shared memory operations.	shmsop(2)
shmget: get	shared memory segment.	shmget(2)
system: issue a	shell command from Fortran.	system(3F)
system: issue a	shell command.	system(3S)
shl:	shell layer manager.	shl(1)
accounting. chargefee:	shell procedures for	acctsh(1M)
addhd:	shell script for add-on disk drive	addhd(1M)
brc: system initialization	shell scripts.	brc(1M)
command programming/ sh:	shell, the standard/restricted	sh(1)
	shl: shell layer manager.	shl(1)
operations.	shmctl: shared memory control	shmctl(2)
segment.	shmget: get shared memory	shmget(2)
operations.	shmop: shared memory	shmop(2)
processing.	shutdown: terminate all	shutdown(1M)
program. sdiff:	side-by-side difference	sdiff(1)
intrinsic function.	sign: Fortran transfer-of-sign	sign(3F)
login:	sign on.	login(1)
pause: suspend process until	signal.	pause(2)
what to do upon receipt of a	signal. signal: specify	signal(2)
action on receipt of a system	signal. /specify Fortran	signal(3F)
on receipt of a system/	signal: specify Fortran action	signal(3F)
upon receipt of a signal.	signal: specify what to do	signal(2)
of processes. kill: send a	signal to a process or a group	kill(2)
ssignal: software	signals.	ssignal(3C)
lex: generate programs for	simple lexical tasks.	lex(1)
generator. rand:	simple random-number	rand(3C)
function.	sin: Fortran sine intrinsic	sin(3F)
	sin: trigonometric functions.	trig(3M)
sin: Fortran	sine intrinsic function.	sin(3F)
sinh: Fortran hyperbolic	sine intrinsic function.	sinh(3F)
intrinsic function.	sinh: Fortran hyperbolic sine	sinh(3F)
	sinh: hyperbolic functions.	sinh(3M)
common object files.	size: print section sizes of	size(1)
size: print section	sizes of common object files.	size(1)
an interval.	sleep: suspend execution for	sleep(1)
interval.	sleep: suspend execution for	sleep(3C)
current/ ttyslot: find the	slot in the utmp file of the	ttyslot(3C)
spline: interpolate	smooth curve.	spline(1G)
	sno: SNOBOL interpreter.	sno(1)
sno:	SNOBOL interpreter.	sno(1)
pg: file perusal filter for	soft-copy terminals.	pg(1)
ssignal:	software signals.	ssignal(3C)

sort:	sort and/or merge files.	sort(1)
qsort: quicker	sort.	qsort(3C)
	sort: sort and/or merge files.	sort(1)
tsort: topological	sort.	tsort(1)
or reject lines common to two	sorted files. comm: select	comm(1)
bsearch: binary search a	sorted table.	bsearch(3C)
brk: change data segment	space allocation.	brk(2)
terminal. ct:	spawn getty to a remote	ct(1C)
fspec: format	specification in text files.	fspec(4)
receipt of a system/ signal:	specify Fortran action on	signal(3F)
receipt of a signal. signal:	specify what to do upon	signal(2)
/set terminal type. modes,	speed, and line discipline.	getty(1M)
used by getty. gettydefs:	speed and terminal settings	gettydefs(4)
	spell: find spelling errors.	spell(1)
spell: find	spelling errors.	spell(1)
curve.	spline: interpolate smooth	spline(1G)
split:	split a file into pieces.	split(1)
csplit: context	split.	csplit(1)
files. fsplit:	split f77, ratfor, or efl	fsplit(1)
pieces.	split: split a file into	split(1)
uuclean: uucp	spool directory clean-up.	uuclean(1M)
lpadmin: configure the LP	spooling system.	lpadmin(1M)
data in a machine-independent/	sputl: access long integer	sputl(3X)
intrinsic function.	sqrt: Fortran square root	sqrt(3F)
exponential, logarithm, power,	square root functions. exp:	exp(3M)
function. sqrt: Fortran	square root intrinsic	sqrt(3F)
	ssignal: software signals.	ssignal(3C)
package. stdio:	standard buffered input/output	stdio(3S)
communication package. ftok:	standard interprocess	stdipc(3C)
programming/ sh: shell, the	standard/restricted command	sh(1)
scheduler lpsched:	start/stop the LP request	lpsched(1M)
startnet:	starts WorkNet.	startnet(1M)
system call.	stat: data returned by stat	stat(5)
	stat: get file status.	stat(2)
useful with graphical/	stat: statistical network	stat(1G)
stat: data returned by	stat system call.	stat(5)
with graphical/ stat:	statistical network useful	stat(1G)
ff: list file names and	statistics for a file system.	ff(1M)
ustat: get file system	statistics.	ustat(2)
lpstat: print LP	status information.	lpstat(1)
ferror: stream	status inquiries.	ferror(3S)
control. uustat: uucp	status inquiry and job	uustat(1C)
communication facilities	status. /report inter-process	ipcs(1)
ps: report process	status.	ps(1)
stat: get file	status.	stat(2)
input/output package.	stdio: standard buffered	stdio(3S)
	stime: set time.	stime(2)
wait for child process to	stop or terminate. wait:	wait(2)
stopnet:	stops WorkNet gracefully.	stopnet(1M)
	strcat: string operations.	string(3C)
sed:	stream editor.	sed(1)
fclose: close or flush a	stream.	fclose(3S)
fopen: open a	stream.	fopen(3S)
reposition a file pointer in a	stream. fseek:	fseek(3S)
get character or word from a	stream. getc:	getc(3S)
gets: get a string from a	stream.	gets(3S)
put character or word on a	stream. putc:	putc(3S)
puts: put a string on a	stream.	puts(3S)
setbuf: assign buffering to a	stream.	setbuf(3S)
ferror:	stream status inquiries.	ferror(3S)

	tabs: set tabs on a terminal . . .	tabs(1)
a file.	tail: deliver the last part of . .	tail(1)
function.	tan: Fortran tangent intrinsic . .	tan(3F)
tan: Fortran	tangent intrinsic function. . . .	tan(3F)
tanh: Fortran hyperbolic	tangent intrinsic function. . . .	tanh(3F)
tangent intrinsic function.	tanh: Fortran hyperbolic	tanh(3F)
gt: general driver for	tape drives.	gt(7)
archive:	tape file archiver.	archive(1)
tar:	tape file archiver.	tar(1)
recover:	tape file restorer.	recover(1)
recover files from a backup	tape. frec:	frec(1M)
	tar: tape file archiver.	tar(1)
programs for simple lexical	tasks. lex: generate	lex(1)
	tee: pipe fitting.	tee(1)
4014: paginator for the	TEXTRONIX 4014 terminal.	4014(1)
tmpfile: create a	temporary file.	tmpfile(3S)
tmpnam: create a name for a	temporary file.	tmpnam(3S)
terminals.	term: conventional names for . . .	term(5)
term: format of compiled	term file.	term(4)
file..	term: format of compiled term . . .	term(4)
for the TEKTRONIX 4014	terminal. 4014: paginator	4014(1)
functions of the DASI 450	terminal. 450: handle special . . .	450(1)
terminfo:	terminal capability data base. . .	terminfo(4)
ct: spawn getty to a remote	terminal.	ct(1C)
generate file name for	terminal. ctermid:	ctermid(3S)
greek: select	terminal filter.	greek(1)
termio: general	terminal interface.	termio(7)
tty: controlling	terminal interface.	tty(7)
dial: establish an out-going	terminal line connection.	dial(3C)
getty. gettydefs: speed and	terminal settings used by	gettydefs(4)
stty: set the options for a	terminal.	stty(1)
tabs: set tabs on a	terminal.	tabs(1)
tty: get the name of the	terminal.	tty(1)
ttyname: find name of a	terminal.	ttyname(3C)
and line/ getty: set	terminal type, modes, speed, . . .	getty(1M)
300: DASI 300 and 300s	terminals.	300(1)
handle special functions of HP	terminals. hp:	hp(1)
perusal filter for soft-copy	terminals. pg: file	pg(1)
term: conventional names for	terminals.	term(5)
kill:	terminate a process.	kill(1)
shutdown:	terminate all processing.	shutdown(1M)
abort:	terminate Fortran program.	abort(3F)
exit:	terminate process.	exit(2)
daemon. errstop:	terminate the error-logging	errstop(1M)
for child process to stop or	terminate. wait: wait	wait(2)
tic:	terminfo compiler.	tic(1M)
tput: query	terminfo database.	tput(1)
data base.	terminfo: terminal capability . . .	terminfo(4)
interface.	termio: general terminal	termio(7)
command.	test: condition evaluation	test(1)
ed, red:	text editor.	ed(1)
edit:	text editor	edit(1)
ex:	text editor.	ex(1)
change the format of a	text file. newform:	newform(1)
fspec: format specification in	text files.	fspec(4)
plock: lock process,	text, or data in memory.	plock(2)
	tic: terminfo compiler.	tic(1M)
date and system/ timex:	time a command; report process . .	timex(1)
time:	time a command.	time(1)
mclock: return Fortran	time accounting.	mclock(3F)

execute commands at a later	time. at:	at(1)
systems for optimal access	time. dcopy: file	dcopy(1M)
	time: get time.	time(2)
profil: execution	time profile.	profil(2)
up an environment at login	time. profile: setting	profile(4)
stime: set	time.	stime(2)
	time: time a command.	time(1)
time: get	time.	time(2)
ctime: convert date and	time to string.	ctime(3C)
clock: report CPU	time used.	clock(3C)
process times.	times: get process and child . . .	times(2)
update access and modification	times of a file. touch:	touch(1)
get process and child process	times. times:	times(2)
file access and modification	times. utime: set	utime(2)
process data and system/	timex: time a command; report . . .	timex(1)
file.	tmpfile: create a temporary	tmpfile(3S)
temporary file.	tmpnam: create a name for a	tmpnam(3S)
contents routines.	toc: graphical table of	toc(1G)
popen: initiate pipe	to/from a process.	popen(3S)
tsort:	topological sort.	tstor(1)
acctmerg: merge or add	total accounting files.	acctmerg(1M)
modification times of a file.	touch: update access and	touch(1)
	toupper: translate characters. . .	conv(3C)
	tplot: graphics filters.	tplot(1G)
	tput: query terminfo database. . .	tput(1)
	tr: translate characters.	tr(1)
	trace: event-tracing driver. . . .	trace(7)
ptrace: process	trace.	ptrace(2)
function. sign: Fortran	transfer-of-sign intrinsic	sign(3F)
toupper:	translate characters.	conv(3C)
tr:	translate characters.	tr(1)
ftw: walk a file	tree.	ftw(3C)
tsearch: manage binary search	trees.	tsearch(3C)
report.	trenter: enter a trouble	trenter(1)
sin:	trigonometric functions.	trig(3M)
trenter: enter a	trouble report.	trenter(1)
	true: provide truth values. . . .	true(1)
type. provide	truth value about processor	machid(1)
true: provide	truth values.	true(1)
trees.	tsearch: manage binary search . . .	tsearch(3C)
	tsort: topological sort.	tsort(1)
interface.	tty: controlling terminal	tty(7)
terminal.	tty: get the name of the	tty(1)
terminal.	ttyname: find name of a	ttyname(3C)
utmp file of the current/	ttyslot: find the slot in the . . .	ttyslot(3C)
int: explicit Fortran	type conversion.	ftype(3F)
file: determine file	type.	file(1)
truth value about processor	type. provide	machid(1)
getty: set terminal	type, modes, speed, and line/ . . .	getty(1M)
types.	types: primitive system data . . .	types(5)
types: primitive system data	types.	types(5)
getpw: get name from	UID.	getpw(3C)
limits.	ulimit: get and set user	ulimit(2)
creation mask.	umask: set and get file	umask(2)
mask.	umask: set file-creation mode . . .	umask(1)
	umount: unmount a file system. . .	umount(2)
UNIX system.	uname: get name of current	uname(2)
UNIX system.	uname: print home of current . . .	uname(1)
file. unget:	undo a previous get of an SCCS . .	unget(1)
an SCCS file.	unget: undo a previous get of . . .	unget(1)

into input stream.	ungetc: push character back	ungetc(3S)
drand48: generate	uniformly distributed/	drand48(3C)
a file.	uniq: report repeated lines in . .	uniq(1)
mktemp: make a	unique file name.	mktemp(3C)
acu: Automatic Call	Unit (ACU) interface.	acu(7)
	units: conversion program.	units(1)
and boot procedures.	boot: UNIX system startup	boot(8)
execution. uux:	UNIX-to-UNIX system command	uux(1CF)
uuto: public	UNIX-to-UNIX system file copy. . .	uuto(1C)
entry.	unlink: remove directory	unlink(2)
link: exercise link and	unlink system calls.	link(1M)
umount:	unmount a file system.	umount(2)
times of a file. touch:	update access and modification . .	touch(1)
of programs. make: maintain,	update, and regenerate groups . . .	make(1)
bdblk: print, initialize,	update bad information	bdblk(1M)
lsearch: linear search and	update.	lsearch(3C)
sync:	update super-block.	sync(2)
sync:	update the super block.	sync(1)
du: summarize disk	usage.	du(1)
stat: statistical network	useful with graphical/	stat(1G)
id: print	user and group IDs and names. . .	id(1)
setuid, setgid: set	user and group IDs.	setuid(2)
crontab:	user crontab file.	crontab(1)
character login name of the	user. cuserid: get	cuserid(3S)
environ:	user environment.	environ(5)
disk accounting data by	user ID. diskusg: generate	diskusg(1M)
getuid: get	user IDs	getuid(2)
ulimit: get and set	user limits.	ulimit(2)
logname: return login name of	user.	logname(3X)
become super-user or another	user. su:	su(1)
the utmp file of the current	user. /find the slot in	ttyslot(3C)
write: write to another	user.	write(1)
mail: send mail to	users or read mail.	mail(1)
wall: write to all	users.	wall(1M)
fuser: identify processes	using a file or file/	fuser(1M)
statistics.	ustat: get file system	ustat(2)
gutil: graphical	utilities.	gutil(1G)
modification times.	utime: set file access and	utime(2)
utmp:	utmp and wtmp entry formats. . . .	utmp(4)
getutent: access	utemp file entry.	getut(3C)
ttyslot: find the slot in the	utmp file of the current user. . .	ttyslot(3C)
formats.	utmp: utmp and wtmp entry	utmp(4)
clean-up.	uuclean: uucp spool directory . . .	uuclean(1M)
uusub: monitor	uucp network.	uusub(1M)
uuclean:	uucp spool directory clean-up. . .	uuclean(1M)
control. uustat:	uucp status inquiry and job	uustat(1C)
system copy.	uucp: UNIX system to UNIX	uucp(1C)
and job control.	uustat: uucp status inquiry	uustat(1C)
	uusub: monitor uucp network. . . .	uusub(1M)
system file copy.	uuto: public UNIX-to-UNIX	uuto(1C)
command execution.	uux: UNIX-to-UNIX system	uux(1C)
	val: validate SCCS file.	val(1)
val:	validate SCCS file.	val(1)
provide truth	value about processor type. . . .	machid(1)
abs: return integer absolute	value.	abs(3C)
abs: Fortran absolute	value.	abs(3F)
getenv: return	value for environment name. . . .	getenv(3C)
ceiling, remainder, absolute	value functions. /floor, ...	floor(3M)
putenv: change or add	value to environment.	putenv(3C)
values.	values: machine-dependent	values(5)

true: provide truth	values.	true(1)
values: machine-dependent	values.	values(5)
/print formatted output of a	varargs argument list.	vprintf(3S)
/print formatted output of a	varargs argument list.	vprintf(3X)
argument list.	varargs: handle variable	varargs(5)
varargs: handle	variable argument list.	varargs(5)
return Fortran environment	variable. getenv:	getenv(3F)
	vc: version control.	vc(1)
option letter from argument	vector. getopt: get	getopt(3C)
assert:	verify program assertion.	assert(3X)
vc:	version control.	vc(1)
get: get a	version of an SCCS file.	get(1)
sccsdiff: compare two	versions of an SCCS file.	sccsdiff(1)
	vi: screen-oriented editor	vi(1)
with label checking.	volcopy: copy file systems	volcopy(1M)
file system: format of system	volume.	fs(4)
output of a varargs argument/	vprintf: print formatted	vprintf(3S)
output of a varargs argument/	vprintf: print formatted	vprintf(3X)
process.	wait: await completion of	wait(1)
or terminate. wait:	wait for child process to stop . .	wait(2)
to stop or terminate.	wait: wait for child process . . .	wait(2)
ftw:	walk a file tree.	ftw(3C)
	wall: write to all users.	wall(1M)
	wc: word count.	wc(1)
	what: identify SCCS files.	what(1)
signal. signal: specify	what to do upon receipt of a . . .	signal(2)
crashes. crash:	what to do when the system	crash(8)
whodo:	who is doing what.	whodo(1M)
who:	who is on the system.	who(1)
	who: who is on the system.	who(1)
	whodo: who is doing what.	whodo(1M)
cd: change	working directory.	cd(1)
chdir: change	working directory.	chdir(2)
get path-name of current	working directory. getcwd:	getcwd(3C)
pwd:	working directory name.	pwd(1)
write:	write on a file.	write(2)
putpwent:	write password file entry.	putpwent(3C)
wall:	write to all users.	wall(1M)
write:	write to another user.	write(1)
	write: write on a file.	write(2)
	write: write to another user. . .	write(1)
open: open for reading or	writing.	open(2)
utmp: utmp and	wtmp entry formats.	utmp(4)
list(s) and execute command.	xargs: construct argument	xargs(1)
functions. and, or,	xor, not: Fortran boolean	bool(3F)
j0, j1, jn,	y0, y1, yn: Bessel functions. . .	bessel(3M)
j0, j1, jn, y0,	y1, yn: Bessel functions.	bessel(3M)
compiler-compiler.	yaac: yet another	yaac(1)
jo, j1, jn, y0, y1,	yn: Bessel functions.	bessel(3M)

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) in the *UNIX System V Programmer Reference Manual.*
How to Get Started, at the front of this volume.

DIAGNOSTICS

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

BUGS

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

WARNINGS

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

NAME

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

SYNOPSIS

300 [**+12**] [**−n**] [**−dt,l,c**]

300s [**+12**] [**−n**] [**−dt,l,c**]

DESCRIPTION

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

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

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

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

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

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

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

Note that the delay control interacts heavily with the prevailing carriage return and line-feed delays. The *stty*(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

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

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

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

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

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

SEE ALSO

450(1), eqn(1), graph(1G), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G).
greek(5) in the *UNIX System V Programmer Reference Manual.*

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.
If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

NAME

4014 – paginator for the TEKTRONIX 4014 terminal

SYNOPSIS

4014 [**−t**] [**−n**] [**−c**N] [**−p**L] [file]

DESCRIPTION

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

The command line options are:

−t Do not wait between pages (useful for directing output into a file).

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

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

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

SEE ALSO

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

NAME

450 − handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

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

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

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

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

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

```
nroff −T450 files ...
```

or

```
nroff −T450−12 files ...
```

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

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).
greek(5) in the *UNIX System V Programmer Reference Manual.*

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.
If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

NAME

acctcom − search and print process accounting file(s)

SYNOPSIS

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

DESCRIPTION

Acctcom reads *file*, the standard input, or **/usr/adm/pacct**, in the form described by *acct*(4) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the **COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE(K)**, and optionally, **F** (the *fork/exec* flag: **1** for *fork* without *exec*), **STAT** (the system exit status), **HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD**, and **BLOCKS /WD** (total blocks read and written).

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

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

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

−a	Show some average statistics about the processes selected. The statistics will be printed after the output records.
−b	Read backwards, showing latest commands first. This *option* has no effect when the standard input is read.
−f	Print the *fork/exec* flag and system exit status columns in the output.
−h	Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as: (total CPU time)/(elapsed time).
−i	Print columns containing the I/O counts in the output.
−k	Instead of memory size, show total kcore-minutes.
−m	Show mean core size (the default).
−r	Show CPU factor (user time/(system-time + user-time).
−t	Show separate system and user CPU times.
−v	Exclude column headings from the output.
−l *line*	Show only processes belonging to terminal **/dev/***line*.
−u *user*	Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a # which designates only those processes executed with *super-user* privileges, or ? which designates only those processes associated with unknown user IDs.
−g *group*	Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.
−s *time*	Select processes existing at or after *time*, given in the format *hr*[:*min*[:*sec*]].
−e *time*	Select processes existing at or before *time*.
−S *time*	Select processes starting at or after *time*.
−E *time*	Select processes ending at or before *time*. Using the same *time* for both **−S** and **−E** shows the processes that existed at *time*.

−n *pattern*	Show only commands matching *pattern* that may be a regular expression as in *ed*(1) except that **+** means one or more occurrences.
−q	Do not print any output records, just print the average statistics as with the **−a** option.
−o *ofile*	Copy selected process records in the input data format to *ofile*; supress standard output printing.
−H *factor*	Show only processes that exceed *factor*, where factor is the "hog factor" as explained in option **−h** above.
−O *sec*	Show only processes with CPU system time exceeding *sec* seconds.
−C *sec*	Show only processes with total CPU time, system plus user, exceeding *sec* seconds.
−I *chars*	Show only processes transferring more characters than the cut-off number given by *chars*.

FILES

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

SEE ALSO

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

BUGS

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.

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** flag is present then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using *adb*. *Adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

In general requests to *adb* are of the form

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

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

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

EXPRESSIONS

. The value of *dot*.

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

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

" The last *address* typed.

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

integer.fraction
A 32-bit floating point number.

'cccc' The ASCII value of up to 4 characters. 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*.

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

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

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

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

Monadic operators:

**exp*	The contents of the location addressed by *exp* in *corfil*.
@*exp*	The contents of the location addressed by *exp* in *objfil*.
−*exp*	Integer negation.
~*exp*	Bitwise complement.

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

e1 +*e2*	Integer addition.
e1 −*e2*	Integer subtraction.
*e1***e2*	Integer multiplication.
e1 %*e2*	Integer division.
e1 & *e2*	Bitwise conjunction.
e1 | *e2*	Bitwise disjunction.
e1#*e2*	*E1* rounded up to the next multiple of *e2*.

COMMANDS

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

?*f*	Locations starting at *address* in *objfil* are printed according to the format *f* and *dot* is incremented by the sum of the increments for each format letter (q.v.).
/*f*	Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is incremented as for ?.
=*f*	The value of *address* itself is printed in the styles indicated by the format *f*. (For **i** format ? is printed for the parts of the instruction that reference subsequent words.)

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

o	2	Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
O	4	Print 4 bytes in octal.
q	2	Print in signed octal.
Q	4	Print long signed octal.
d	2	Print in decimal.

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

new-line

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

[?/]**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?** will cause **/** to refer to *objfil*.)

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

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

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

<*f*	Read commands from the file *f* and return.
>*f*	Send output to the file *f*, which is created if it does not exist.
r	Print the general registers and the instruction addressed by **pc**. *Dot* is set to **pc**.
f	Print the floating registers in single or double length. If the floating point status of **ps** is set to double (0200 bit) then double length is used anyway.
b	Print all breakpoints and their associated counts and commands.
a	ALGOL 68 stack backtrace. If *address* is given then it is taken to be the address of the current frame (instead of **r4**). If *count* is given then only the first *count* frames are printed.
c	C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of **r5**). If **C** is used then the names and (16-bit) values of all automatic and static variables are printed for each active function. If *count* is given then only the first *count* frames are printed.
e	The names and values of external variables are printed.
w	Set the page width for output to *address* (default 80).
s	Set the limit for symbol matches to *address* (default 255).
o	All integers input are regarded as octal.
d	Reset integer input as described in *EXPRESSIONS*.
q	Exit from *adb*.
v	Print all non-zero variables in octal.
m	Print the address map.

:*modifier*
Manage a subprocess. Available modifiers are:

b*c*	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 on entry to the subprocess.
c*s*	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**.

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

VARIABLES

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

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

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

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

ADDRESSES

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

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

otherwise

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

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

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

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

FILES

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

SEE ALSO

ptrace(2), a.out(4), core(4) in the *UNIX System V Programmer Reference Manual.*

DIAGNOSTICS

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

BUGS

A breakpoint set at the entry point is not effective on initial entry to the program.
When single stepping, system calls do not count as an executed instruction.
Local variables whose names are the same as an external variable may foul up the accessing of the external.

NAME

admin − create and administer SCCS files

SYNOPSIS

admin [**−n**] [**−i**[name]] [**−r**rel] [**−t**[name]] [**−f**flag[flag-val]] [**−d**flag[flag-val]] [**−a**login] [**−e**login] [**−m**[mrlist]] [**−y**[comment]] [**−h**] [**−z**] files

DESCRIPTION

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

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

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

−n This keyletter indicates that a new SCCS file is to be created.

−i[*name*] The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see **−r** keyletter for delta numbering scheme). If the **i** keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an *admin* command on which the **i** keyletter is supplied. Using a single *admin* to create two or more SCCS files requires that they be created empty (no **−i** keyletter). Note that the **−i** keyletter implies the **−n** keyletter.

−r*rel* The *rel*ease into which the initial delta is inserted. This keyletter may be used only if the **−i** keyletter is also used. If the **−r** keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

−t[*name*] The *name* of a file from which descriptive text for the SCCS file is to be taken. If the **−t** keyletter is used and *admin* is creating a new SCCS file (the **−n** and/or **−i** keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a **−t** keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a **−t** keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

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

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

c*ceil* The highest release (i.e., "ceiling"), a number less than or equal to 9999, which may be retrieved by a *get*(1) command for editing. The default value for an unspecified **c** flag is 9999.

f*floor* The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a *get*(1) command for editing. The default value for an unspecified **f** flag is 1.

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

i[*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. This allows multiple concurrent updates to the same version of the SCCS file.

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

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

The character **a** in the *list* is equivalent to specifying *all releases* for the named SCCS file.

n Causes *delta*(1) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

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

m*mod* *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, the value assigned 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 value specifies the name of an *MR* number validity checking program (see *delta*(1)). (If this flag is set when creating an SCCS file, the **m** keyletter must also be used even if its value is null).

−d*flag* Causes removal (deletion) of the specified *flag* from an SCCS file. The **−d** keyletter may be specified only when processing existing SCCS files. Several **−d** keyletters may be supplied on a single *admin* command. See the **−f** keyletter for allowable *flag* names.

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

−a*login* A *login* name, or numerical UNIX system group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **a** keyletters may be used on a single *admin* command line. As many *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.

−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** keyletters may be used on a single *admin* command line.

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

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

The **−y** keyletter is valid only if the **−i** and/or **−n** keyletters are specified (i.e., a new SCCS file is being created).

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

−h Causes *admin* to check the structure of the SCCS file (see *sccsfile*(5)), and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file.

Appropriate error diagnostics are produced.

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

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

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

FILES

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

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

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

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

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1).
sccsfile(4) in the *UNIX System V Programmer Reference Manual*.

Source Code Control System User Guide in the *UNIX System V User Guide*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ar − archive and library maintainer for portable archives

SYNOPSIS

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

DESCRIPTION

The *Ar* command 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. The magic string and the file headers used by *ar* consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

When *ar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in *ar*(4). The archive symbol table (described in *ar*(4)) is used by the link editor (*ld*(1)) to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by *ar* when there is at least one object file in the archive. The archive symbol table is in a specially named file which is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the *ar*(1) command is used to create or update the contents of such an archive, the symbol table is rebuilt. The **s** option described below will force the symbol table to be rebuilt.

Key is an optional **−**, followed by 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 dates of modification later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.

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

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

p Print the named files in the archive.

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

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

v Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, give a long listing of all information about the files. When used with **x**, precede each file with a name.

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

l Place temporary files in the local current working directory, rather than in the directory specified by the environment variable **TMPDIR** or in the default directory **/tmp**.

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

FILES

/tmp/ar*	temporaries

SEE ALSO

ld(1), lorder(1), strip(1).
tmpnam(3S), a.out(4), ar(4) in the *UNIX System V Programmer Reference Manual.*

BUGS

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

NAME

archive—saves the contents of a file system to tape

SYNOPSIS

archive [—i string] file system mag tape

DESCRIPTION

Use *archive* to copy the contents of the hard disk to a cartridge tape. If the files will not fit on a single tape, you will be prompted to install a new tape. If possible, run *archive* on an unmounted file system.

Other users must be logged out when you back up /dev/root.

Be sure to specify /dev/rsct (for streaming tape) when running *archive*.

The **i** option puts any character string you specify (up to 128 characters) into the header block on the tape.

EXAMPLES

/etc/dump.hd

This command backs up the first hard disk to tape.

To restore the first hard disk from tape, use the *recover* command.

/etc/mount /dev/hdla
archive /dev/rhdla /dev/rsct

This command backs up the second hard disk to tape.

SEE ALSO

recover(1)

NAME

as − common assembler

SYNOPSIS

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

DESCRIPTION

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

−o *objfile*	Put the output of the assembly in *objfile*. By default, the output file name is formed by removing the **.s** suffix, if there is one, from the input file name and appending a **.o** suffix.
−n	Turn off long/short address optimization. By default, address optimization takes place.
−j	Invoke the long-jump assembler (for the VAX version of the common assembler only). The address optimization algorithm chooses between long and short address lengths, with short lengths chosen when possible. Often, three distinct lengths are allowed by the machine architecture; a choice must be made between two of those lengths. When the two choices given to the assembler exclude the largest length allowed, then some addresses might be unrepresentable. The long-jump assembler will always have the largest length as one of its allowable choices. If the assembler is invoked without this option, and the case arises where an address is unrepresentable by either of the two allowed choices, then the user will be informed of the error, and advised to try again using the **−j** option.
−m	Run the *m4* macro pre-processor on the input to the assembler.
−R	Remove (unlink) the input file after assembly is completed.
−r	Place all assembled data (normally placed in the **.data** section) into the **.text** section (for the VAX version of the common assembler only). This option effectively disables the **.data** pseudo operation. This option is off by default.
−[bwl]	Create byte (**b**), halfword (**w**) or long (**l**) displacements for undefined symbols (for the VAX version of the common assembler only). (An undefined symbol is a reference to a symbol whose definition is external to the input file or a forward reference.) The default value for this option is long (**l**) displacements.
−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) in the *UNIX System V Programmer Reference Manual.*

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 since *m4* cannot determine which are assembler symbols and which are real *m4* macros.

Use the **−b** or **−w** option only when undefined symbols are known to refer to locations representable by the specified default displacement. Use of either option when assembling a file containing a reference to a symbol that is to be resolved by the loader can lead to unpredictable results, since the loader may be unable to place the address of the symbol into the space provided.

BUGS

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

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

NAME

asa — interpret ASA carriage control characters

SYNOPSIS

asa [files]

DESCRIPTION

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

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

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

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

```
a.out | asa | lp
```

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

```
asa file
```

SEE ALSO

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

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* will execute when system load level permits. *At* **-r** removes jobs previously scheduled with *at*. The **-l** option reports all jobs scheduled for the invoking user.

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

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

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

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

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

Thus legitimate commands include:

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

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

Batch submits 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" will respond with the error message too late.

At **-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 also get job numbers by typing *at* **-l**. You can only remove your own jobs unless you are the super-user.

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *Sh*(1) provides 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 depress 'D')
```

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

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

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

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

FILES

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

SEE ALSO

kill(1), mail(1), nice(1), ps(1), sh(1).
cron(1M) in the *UNIX System V Administrator Reference Manual.*

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk − pattern scanning and processing language

SYNOPSIS

awk [−Fc] [prog] [parameters] [files]

DESCRIPTION

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

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

Files are read in order; if there are no files, the standard input is read. The file name − means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS; see below). The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

A pattern-action statement has the form:

```
pattern { action }
```

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

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

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators +, −, *, /, %, and concatenation (indicated by a blank). The C operators ++, −−, +=, −=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").

The *print* statement prints its arguments on the standard output (or on a file if >*expr* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the format (see *printf*(3S)).

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp*, *log*, *sqrt*, and *int*. The last truncates its argument to an integer; *substr*(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf*(*fmt*, *expr*, *expr*, ...) formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (!, | |, **& &**, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep* (see *grep*(1)). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
```

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

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

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

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

or by using the **−F***c* option.

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

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

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

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

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

Print file, filling in page numbers starting at 5:

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

command line: awk −f program n=5 input

SEE ALSO

grep(1), lex(1), sed(1).
malloc(3X) in the *UNIX System V Programmer Reference Manual.*

UNIX System V Support Tools Guide.

BUGS

Input white space is not preserved on output if fields are involved.
There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

NAME

banner — make posters

SYNOPSIS

banner strings

DESCRIPTION

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

SEE ALSO

echo(1).

NAME

basename, dirname — deliver portions of path names

SYNOPSIS

basename string [suffix]
dirname string

DESCRIPTION

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

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

EXAMPLES

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

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

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

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

SEE ALSO

sh(1).

BUGS

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

NAME

bc − arbitrary-precision arithmetic language

SYNOPSIS

bc [**−c**] [**−l**] [file ...]

DESCRIPTION

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

Comments
    are enclosed in /* and */.

Names
    simple variables: L
    array elements: L [E]
    The words "ibase", "obase", and "scale"

Other operands

```
arbitrarily long numbers with optional sign and decimal point.
( E )
sqrt ( E )
length ( E )     number of significant decimal digits
scale ( E )      number of digits right of decimal point
L ( E , ... , E )
```

Operators

```
+ − * / % ^ (% is remainder; ^ is power)
++  −−        (prefix and postfix; apply to names)
== <= >= != < >
= =+ =− =* =/ =% =^
```

Statements

```
E
{ S ; ... ; S }
if ( E ) S
while ( E ) S
for ( E ; E ; E ) S
null statement
break
quit
```

Function definitions

```
define L ( L ,..., L ) {
        auto L, ... , L
        S; ... S
        return ( E )
}
```

Functions in **−l** math library

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

All function arguments are passed by value.

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

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

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

EXAMPLE

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

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

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

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

FILES

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

SEE ALSO

dc(1).

UNIX System V Programmer Guide.

BUGS

No **&&**, **||** yet.
For statement must have all three E's.
Quit is interpreted when read, not when executed.

NAME

bdiff − big diff

SYNOPSIS

bdiff file1 file2 [n] [**−s**]

DESCRIPTION

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

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

FILES

/tmp/bd?????

SEE ALSO

diff(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

bfs — big file scanner

SYNOPSIS

bfs [−] name

DESCRIPTION

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.

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

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

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

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

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

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

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

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

```
xb/^/ label
```

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

xt *number*

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

xv[*digit*][*spaces*][*value*]

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

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

will all print the first 100 lines.

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

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

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

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

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

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

```
xv7\!date
```

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

xbz *label*

xbn *label*

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

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

xc [*switch*]

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

SEE ALSO

csplit(1), ed(1).
regcmp(3X) in the *UNIX System V Programmer Reference Manual.*

DIAGNOSTICS

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

NAME

bs — a compiler/interpreter for modest-sized programs

SYNOPSIS

bs [file [args]]

DESCRIPTION

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

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

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

```
statement
label  statement
```

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

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

Statement Syntax:

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

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

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

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

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

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

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

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

for name = expression expression statement
for name = expression expression
...
next

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

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

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

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

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

if expression statement
if expression
...
[**else**
...]
fi
The statement (first form) or group of statements (second form) is executed if the expression evaluates to non-zero. The strings **0** and "" (null) evaluate as zero. In the second form, an optional *else* allows for a group of statements to be executed when the first group is not. The only statement permitted on the same line with an *else* is an *if*; only other *fi*'s can be on the same line with a *fi*. The elision of *else* and *if* into an *elif* is supported. Only a single *fi* is required to close an *if* ... *elif* ... [*else* ...] sequence.

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

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

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

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

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

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

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

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

! shell command
: An immediate escape to the shell.

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

Expression Syntax:

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

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

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

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

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

(expression)
: Parentheses are used to alter the normal order of evaluation.

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

```
( False, True )[ a == b ]
```

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

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

− expression
: The result is the negation of the expression.

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

−− name
: Decrements the value of the variable. The result is the new value.

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

expression *operator* expression
: Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the function is applied.

Binary Operators (in increasing precedence):

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

_
: _ (underscore) is the concatenation operator.

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

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

+ −
: Add and subtract.

*** / %**
: Multiply, divide, and remainder.

^
: Exponentiation.

Built-in Functions:

Dealing with arguments

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

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

Mathematical

abs(x)
: is the absolute value of *x*.

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

ceil(x)
: returns the smallest integer not less than *x*.

cos(x)
: is the cosine of *x* (radians).

exp(x)
: is the exponential function of *x*.

floor(x)
: returns the largest integer not greater than *x*.

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

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

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

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

String operations

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

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

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

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

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

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

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

File handling

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

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

Examples are given in the following section.

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

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

Tables

table(name, size)

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

item(name, i)

key()

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

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

iskey(name, word)

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

Odds and ends

eval(s)

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

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

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

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

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

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

plot(request, args)

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

Call	*Function*
plot(0, term)	causes further *plot* output to be piped 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()
in immediate mode, *last* returns the most recently computed value.

PROGRAMMING TIPS

Using *bs* as a calculator:

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

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

The outline of a typical *bs* program:

```
# initialize things:
var1 = 1
open("read", "infile", "r")
...
# compute:
```

```
while  ?(str = read)
        ...
next
# clean up:
close("read")
...
# last statement executed (exit or stop):
exit
# last input line:
run
```

Input/Output examples:

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

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

SEE ALSO

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

NAME

cal − print calendar

SYNOPSIS

cal [[month] year]

DESCRIPTION

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

Try September 1752.

BUGS

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

NAME

calendar − reminder service

SYNOPSIS

calendar [−]

DESCRIPTION

Calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Aug. 24," "august 24," "8/24," etc., are recognized, but not "24 August" or "24/8". On weekends "tomorrow" extends through Monday.

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

FILES

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

SEE ALSO

mail(1).

BUGS

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

NAME

cat − concatenate and print files

SYNOPSIS

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

DESCRIPTION

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

```
cat file
```

prints the file, and:

```
cat file1 file2 >file3
```

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

If no input file is given, or if the argument − is encountered, *cat* reads from the standard input file. Output is buffered unless the **−u** option is specified. The **−s** option makes *cat* silent about non-existent files.

The **−v** option causes non-printing characters (with the exception of tabs, new-lines and form-feeds) to be printed visibly. Control characters are printed ^*X* (control-*x*); the DEL character (octal 0177) is printed **^?**. Non-ASCII characters (with the high bit set) are printed as **M-***x*, where *x* is the character specified by the seven low order bits.

When used with the **−v** option, **−t** causes tabs to be printed as **^I**'s, and **−e** causes a **$** character to be printed at the end of each line (prior to the new-line). The **−t** and **−e** options are ignored if the **−v** option is not specified.

WARNING

Command formats such as

```
cat file1 file2 >file1
```

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

SEE ALSO

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

NAME

cb − C program beautifier

SYNOPSIS

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

DESCRIPTION

Cb reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code. Under default options, *cb* preserves all user newlines. Under the **−s** flag *cb* canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*. The **−j** flag causes split lines to be put back together. The **−l** flag causes *cb* to split lines that are longer than *leng*.

SEE ALSO

cc(1).

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

BUGS

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

NAME

cc, pcc − C compiler

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

−p Arrange for the compiler to produce code 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 arranges to write out a **mon.out** file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1). For the PDP-11 only, the libraries **/lib/libp/libm.a** (if the −lm option is used) and **/lib/libp/libc.a** must be specified explicitly if the versions reporting function call counts are to be loaded.

−f Link the object program with the floating-point interpreter for systems without hardware floating-point.

−g Cause the compiler to generate additional information needed for the use of *sdb*(1).

−O Invoke an object-code optimizer.

−S Compile the named C programs and leave the assembler-language output on corresponding files suffixed **.s**.

−E Run only *cpp*(1) on the named C programs and send the result to the standard output.

−P Run only *cpp*(1) on the named C programs and leave the result on corresponding files suffixed **.i**.

−B*string*
Construct path names for substitute preprocessor, compiler, assembler and link editor passes by concatenating *string* with the suffixes **cpp**, **c0** (or **ccom** or **comp**, see under FILES below), **c1**, **c2** (or **optim**), **as** and **ld**. If *string* is empty it is taken to be **/lib/o**.

−t[p012al]
Find only the designated preprocessor, compiler, assembler and link editor passes in the files whose names are constructed by a **−B** option. In the absence of a **−B** option, the *string* is taken to be **/lib/n**. The value **−t** "" is equivalent to **−tp012**.

−W*c,arg1[,arg2...]*
Hand off the argument[s] *argi* to pass *c* where *c* is one of **[p012al]**

indicating preprocessor, compiler first pass, compiler second pass, optimizer, assembler, or link editor, respectively.

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

The C language standard was extended to include arbitrary length variable names. This standard has been implemented on the VAX and the 3B 20 computer, but not on the PDP-11. The option pair "**−Wp,−T −W0,−XT**" will cause the current compiler (on the 3B 20 computer and the VAX) to behave the same as previous compilers with respect to the length of variable names.

FILES

file.c	input file
file.o	object file
a.out	linked output
/tmp/ctm*	temporary
/usr/tmp/ctm*	temporary
/lib/cpp	C preprocessor *cpp*(1)
/lib/c[01]	PDP-11 compiler, *cc*
/usr/lib/comp	compiler, *pcc*
/lib/ccom	VAX compiler, *cc*
/lib/comp	3B 20 computer compiler *cc*
/lib/c2	VAX and PDP-11 optional optimizer
/lib/optim	3B 20 computer optional optimizer
/usr/lib/Oc*	backup compiler, *Occ*
/bin/as	assembler, *as*(1)
/bin/ld	link editor, *ld*(1)
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	profiling startoff
/lib/fcrt0.o	floating-point interpretation startoff (PDP-11)
/lib/fmcrt0.o	floating-point interpretation and profiling startoff (PDP-11)
/lib/libc.a	standard C library, see section (3) in the *UNIX System V Programmer Reference Manual*
/lib/libp/lib*.a	profiled versions of libraries

SEE ALSO

adb(1), cpp(1), as(1), ld(1), prof(1), sdb(1).
exit(2), monitor(3C) in the *UNIX System V Programmer Reference Manual.*

The C Programming Language by B. W. Kernighan.
Programming in C-A Tutorial by B. W. Kernighan.
C Reference Manual by D. M. Ritchie.

NOTES

By default, the return value from a C program is completely random. The only two guaranteed ways to return a specific value are to explicitly call *exit*(2) or to leave the function **main**() with a "**return** *expression*;" construct.

DIAGNOSTICS

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

NAME

cd — change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

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

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

SEE ALSO

pwd(1), sh(1).
chdir(2) in the *UNIX System V Programmer Reference Manual.*

NAME

cdc − change the delta commentary of an SCCS delta

SYNOPSIS

cdc −**r**SID [−**m**[mrlist]] [−**y**[comment]] files

DESCRIPTION

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

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

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

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

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

−**r***SID* Used to specify the *S*CCS *ID*entification (*SID*) string of a delta for which the delta commentary is to be changed.

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

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

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

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

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

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

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

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

EXAMPLES

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

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

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

does the same thing.

WARNINGS

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

FILES

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

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1).
sccsfile(4) in the *UNIX System V Programmer Reference Manual.*

Source Code Control System User Guide in the *UNIX System V User Guide.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

cflow− generate C flowgraph

SYNOPSIS

cflow [**−r**] [**−ix**] [**−i_**] [**−d**num] files

DESCRIPTION

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

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

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

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

```
int     i;

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

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

the command

```
cflow −ix file.c
```

produces the output

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

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

The following options are interpreted by *cflow*:

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

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

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

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

DIAGNOSTICS

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

SEE ALSO

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

BUGS

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

NAME

chmod − change mode

SYNOPSIS

chmod mode files

DESCRIPTION

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

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

A symbolic *mode* has the form:

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

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

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

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

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g** and **t** only works with **u**.

Only the owner of a file (or the super-user) may change its mode. Only the super-user may set the sticky bit. In order to set the group ID, the group of the file must correspond to your current group ID.

EXAMPLES

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

```
chmod o−w file

chmod +x file
```

SEE ALSO

ls(1).
chmod(2) in the *UNIX System V Programmer Reference Manual.*

NAME

chown, chgrp — change owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

DESCRIPTION

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

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

If either command is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1).
chown(2), group(4), passwd(4) in the *UNIX System V Programmer Reference Manual.*

NAME

cmp − compare two files

SYNOPSIS

cmp [**−l**] [**−s**] file1 file2

DESCRIPTION

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

Options:

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

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

SEE ALSO

comm(1), diff(1).

DIAGNOSTICS

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

NAME

col − filter reverse line-feeds

SYNOPSIS

col [−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.

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

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

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

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

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

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

SEE ALSO

nroff(1), tbl(1).

NOTES

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

BUGS

Cannot back up more than 128 lines.
Allows at most 800 characters, including backspaces, on a line.
Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

NAME

comb − combine SCCS deltas

SYNOPSIS

comb [**−o**] [**−s**] [**−psid**] [**−clist**] files

DESCRIPTION

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

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

−p*SID* The *S*CCS *ID*entification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

−c*list* A *list* (see *get*(1) for the syntax of a *list*) of deltas to be preserved. All other deltas are discarded.

−o For each **get −e** generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the **−o** keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

−s This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 * (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 keyletter arguments are specified, *comb* will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

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

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sh(1).
sccsfile(4) in the *UNIX System V Programmer Reference Manual.*

Source Code Control System User Guide in the *UNIX System V User Guide*.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

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

NAME

comm − select or reject lines common to two sorted files

SYNOPSIS

comm [− [**123**]] file1 file2

DESCRIPTION

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

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

SEE ALSO

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

NAME

cp, ln, mv — copy, link or move files

SYNOPSIS

cp file1 [file2 ...] target
ln [**−f**] file1 [file2 ...] target
mv [**−f**] file1 [file2 ...] target

DESCRIPTION

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

If *mv* or *ln* determines that the mode of *target* forbids writing, it will print the mode (see *chmod*(2)), ask for a response, and read the standard input for one line; if the line begins with **y**, the *mv* or *ln* occurs, if permissable; if not, the command exits. 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.

Only *mv* will allow *file1* to be a directory, in which case the directory rename will occur only if the two directories have the same parent; *file1* is renamed *target*. If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

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

SEE ALSO

cpio(1), rm(1).
chmod(2) in the *UNIX System V Programmer Reference Manual.*

BUGS

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

Ln will not link across file systems.

NAME

cpio − copy file archives in and out

SYNOPSIS

cpio −o [**acBv**]

cpio −i [**BcdmrtuvfsSb6**] [patterns]

cpio −p [**adlmruv**] directory

DESCRIPTION

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

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

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

The meanings of the available options are:

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

EXAMPLES

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

```
ls | cpio −o >/dev/mt/0m

cd olddir
find . −depth −print | cpio −pdl newdir
```

The trivial case "find . −depth −print | cpio −oB >/dev/rmt/0m" can be handled more efficiently by:

```
find . −cpio /dev/rmt/0m
```

SEE ALSO

ar(1), find(1), ls(1).
cpio(4) in the *UNIX System V Programmer Reference Manual.*

BUGS

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

NAME

cpp − the C language preprocessor

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

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

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, pdp11, u370, u3b, 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 will be undefined regardless of the order of the options.

−T Except on the PDP-11, 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 "" will be 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.

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

All *cpp* directives start with lines begun by #. 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* has been expanded, *cpp* re-starts its scan for names to expand at the beginning of newly created *token-string*.

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

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

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

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

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

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

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

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

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

FILES

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

SEE ALSO

cc(1), m4(1).

DIAGNOSTICS

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

NOTES

When 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.

NAME

cpu—display the local system name in WorkNet

SYNOPSIS

cpu

DESCRIPTION

Use the *cpu* command to display the name of your local system in the WorkNet network. This command tells you the name of the machine that connects your terminal to WorkNet.

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 users' crontabs. The −r option removes a user's crontab from the crontab directory. *Crontab* −l will list the crontab file for the invoking user.

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

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

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

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

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

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

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 will be mailed to the 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

sh(1).
cron(1M) in the *UNIX System V Administrator Reference Manual.*

NAME

crypt — encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

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

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

will print the clear.

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

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

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

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

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. 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), stty(1).

BUGS

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

NAME

csplit − context split

SYNOPSIS

csplit [**−s**] [**−k**] [**−f** prefix] file arg1 [... argn]

DESCRIPTION

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

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

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

The options to *csplit* are:

−s *Csplit* normally prints the character counts for each file created. If the **−s** option is present, *csplit* suppresses the printing of all character counts.

−k *Csplit* normally removes created files if an error occurs. If the **−k** option is present, *csplit* leaves previously created files intact.

−f *prefix* If the **−f** option is used, the created files are named *prefix***00** ... *prefixn*. The default is **xx00** ... **xx***n*.

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

/rexp/ A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional + or − some number of lines (e.g., **/Page/−5**).

%rexp% This argument is the same as */rexp/*, except that no file is created for the section.

lnno A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.

{*num*} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

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

EXAMPLES

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

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

```
cat cobol0[0−3] > file
```

Note that this example overwrites the original file.

```
csplit −k file 100 {99}
```

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

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

Assuming that **prog.c** follows the normal **C** coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate **C** routine (up to 21) in **prog.c**.

SEE ALSO

ed(1), sh(1).
regexp(5) in the *UNIX System V Programmer Reference Manual.*

DIAGNOSTICS

Self-explanatory except for:

```
arg − out of range
```

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

NAME

ct − spawn getty to a remote terminal

SYNOPSIS

ct [**−h**] [**−v**] [**−wn**] [**−s**speed] telno ...

DESCRIPTION

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

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

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

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

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

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

FILES

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

SEE ALSO

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

NAME

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 will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the *bfs*(1) or *tail*(1) commands.

The only *options* you will 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 also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these *options*:

−o	Octal
−x	Hexadecimal
−u	Unsigned
−e	Floating point

These *options* are used only in special circumstances:

−l *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.

−s Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.

−t *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The Diagnostics section explains when to use this option.

−P Run the C preprocessor on the input before tracing it. You can also use the **-D**, **-I**, and **-U** *cc*(1) preprocessor options.

These *options* are used to tailor the run-time trace package when the traced program will run in a non-UNIX system environment:

−b Use only basic functions in the trace code, that is, those in *ctype*(3C), *printf*(3S), and *string*(3C). These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have *signal*(2), *fflush*(3S), *longjmp*(3C), or *setjmp*(3C).

−p *'s'* Change the trace print function from the default of 'printf('. For example, 'fprintf(stderr,' would send the trace to the standard error output.

-r f Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **-p** option).

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 will be compiled and executed. The output of the program will be the number **2**, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke *ctrace* with these commands: ctrace lc.c >temp.c cc temp.c a.out the output will be:

```
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 will be:

```
        /* 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 char-
```

acter (cntl-d) the final output will be:

```
        /* c == -1 */ 10        printf("%d\n", nl);
        /* nl == 2 */2          return
```

Note that the program output printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator (=) in place of the equal operator (==). You can easily miss this error during code reading.

EXECUTION-TIME TRACE CONTROL

The default operation for *ctrace* is to trace the entire program file, unless you use the **-f** or **-v** options to trace specific functions. This does not give you statement-by-statement control of the tracing, nor does it let you turn the tracing off and on when executing the 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), since the traced code often gets some *cc* warning messages. You can get *cc* error messages in some rare cases, all of which can be avoided.

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 will now get.

redeclaration of signal
: Either correct this declaration of *signal*(2), or remove it and #include <signal.h>.

WARNINGS

You will get a *ctrace* syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

Ctrace assumes that BADMAG is a preprocessor macro, and that EOF and NULL are #defined constants. Declaring any of these to be variables, e.g., "int EOF;", will cause a syntax error.

BUGS

Ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. Ctrace may choose to print the address of an aggregate or use the wrong format (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) in the *UNIX System V Programmer Reference Manual.*

NAME

cu − call another UNIX system

SYNOPSIS

cu [−sspeed] [−lline] [**−h**] [**−t**] [**−d**] [**−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.

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 will search the L-devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made. The specified device is generally a directly connected asynchronous line (e.g., **/dev/ttyab**), in this case a telephone 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 telephone 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.

−d Causes diagnostic traces to be printed.

−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.

−m Designates a direct line which has modem control.

−n Will request the telephone 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 teletelephone number with equal signs for secondary dial tone or minus signs for delays, at appropriate places.

systemname
A **uucp** system name may be used rather than a telephone number; in this case, cu will obtain an appropriate direct line or telephone number from **/usr/lib/uucp/L.sys** (the appropriate baud rate is also read along with telephone numbers). Cu will try each telephone 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 will use the line specified by the -l option.

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 will cause the command to be run by a sub-shell; probably not what was intended.**
~%**take** *from* [*to*]	copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
~%**put** *from* [*to*]	copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.
~~...	send the line ~... to the remote system.
~%**break**	transmit a **BREAK** to the remote system.
~%**nostop**	toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. 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 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 dial a system whose number is 9 201 555 1212 using 1200 baud:

```
cu -s1200 9=2015551212
```

If the speed is not specified, 300 is the default value.

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
```

To dial a system using a specific line:

```
cu -l /dev/culXX 2015551212
```

To use a system name:

```
cu YYYZZZ
```

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.

BUGS

Cu buffers input internally.
There is an artificial slowing of transmission by *cu* during the ~%**put** operation so that loss of data is unlikely.
You cannot use **cu** from the 3B 20 computer system console.

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 parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (−**c** option) or the length can vary from line to line and be marked with a field delimiter character like *tab* (−**f** option). *Cut* can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

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

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

HINTS

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

EXAMPLES

cut −d: −f1,5 /etc/passwd	mapping of user IDs to names
name=`who am i | cut −f1 −d" "`	to set **name** to current login name.

DIAGNOSTICS

line too long	A line can have no more than 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).

NAME

cxref − generate C program cross-reference

SYNOPSIS

cxref [options] files

DESCRIPTION

Cxref analyzes a collection of C files and attempts to build a cross-reference table. *Cxref* utilizes a special version of *cpp* to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the **−c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

In addition to the **−D**, **−I** and **−U** options (which are identical to their interpretation by *cc*(1)), the following *options* are interpreted by *cxref*:

−c Print a combined cross-reference of all input files.

−w<*num*>
Width option which formats output no wider than <num> (decimal) columns. This option will default to 80 if <num> is not specified or is less than 51.

−o file Direct output to named *file*.

−s Operate silently; does not print input file names.

−t Format listing for 80-column width.

FILES

/usr/lib/xcpp special version of C-preprocessor.

SEE ALSO

cc(1).

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files, anyway.

BUGS

Cxref considers a formal argument in a *#define* macro definition to be a declaration of that symbol. For example, a program that *#include*s **ctype.h**, will contain many declarations of the variable **c**.

NAME

date − print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time.

If the argument begins with +, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to *printf*(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year − 01 to 12
d	day of month − 01 to 31
y	last 2 digits of year − 00 to 99
D	date as mm/dd/yy
H	hour − 00 to 23
M	minute − 00 to 59
S	second − 00 to 59
T	time as HH:MM:SS
j	day of year − 001 to 366
w	day of week − Sunday = 0
a	abbreviated weekday − Sun to Sat
h	abbreviated month − Jan to Dec
r	time in AM/PM notation

EXAMPLE

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

would have generated as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

DIAGNOSTICS

No permission	if you are not the super-user and you try to change the date;
bad conversion	if the date set is syntactically incorrect;
bad format character	if the field descriptor is not recognizable.

FILES

/dev/kmem

SEE ALSO

printf(3S) in the *UNIX System V Programmer Reference Manual.*

WARNING

It is a bad practice to change the date while the system is running multi-user.

NAME

dc − desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. (See *bc*(1), a preprocessor for *dc* that provides infix notation and a C-like syntax that implements functions. *Bc* also provides reasonable control structures for programs.) The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number
The value of the number is pushed on the stack. A number is an unbroken string of the digits 0−9. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.

+ − / * % ^
The top two values on the stack are added (+), subtracted (−), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

s*x* The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the **s** is capitalized, *x* is treated as a stack and the value is pushed on it.

l*x* The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d The top value on the stack is duplicated.

p The top value on the stack is printed. The top value remains unchanged. **P** interprets the top of the stack as an ASCII string, removes it, and prints it.

f All values on the stack are printed.

q exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

x treats the top element of the stack as a character string and executes it as a string of *dc* commands.

X replaces the number on the top of the stack with its scale factor.

[...] puts the bracketed ASCII string onto the top of the stack.

<*x* **>***x* **=***x*
The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.

v replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

! interprets the rest of the line as a UNIX system command.

c All values on the stack are popped.

i The top value on the stack is popped and used as the number radix for further input. **I** pushes the input base on the top of the stack.

o The top value on the stack is popped and used as the number radix for further output.

O pushes the output base on the top of the stack.

k the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

z The stack level is pushed onto the stack.

Z replaces the number on the top of the stack with its length.

? A line of input is taken from the input source (usually the terminal) and executed.

; : are used by *bc* for array operations.

EXAMPLE

This example prints the first ten values of n!:

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

SEE ALSO

bc(1).

DIAGNOSTICS

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

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

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

Out of headers
for too many numbers being kept around.

Out of pushdown
for too many items on the stack.

Nesting Depth
for too many levels of nested execution.

NAME

dd − convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

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

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if *ascii* or *ebcdic* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output block of size *cbs*.

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

EXAMPLE

This command will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the ASCII file **x**:

```
dd if=/dev/rmt/0m of=x ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape. *Dd* is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary block sizes.

SEE ALSO

cp(1).

DIAGNOSTICS

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

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256-character standard in the CACM Nov, 1968. The *ibm* conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

New-lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

NAME

delta − make a delta (change) to an SCCS file

SYNOPSIS

delta [**−r**SID] [**−s**] [**−n**] [**−g**list] [**−m**[mrlist]] [**−y**[comment]] [**−p**] files

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

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

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin*(1)) that may be present in the SCCS file (see **−m** and **−y** keyletters below).

Keyletter arguments apply independently to each named file.

−r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *get*s for editing (**get −e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **−r** keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

−s Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

−n Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

−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*] If the SCCS file has the **v** flag set (see *admin*(1)) then a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

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

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

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the **MR** numbers. If a non-zero exit status is returned from **MR** number validation program, *delta* terminates. (It is assumed that the **MR** numbers were not all valid.)

−y[*comment*]	Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*. If **−y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.
−p	Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

All files of the form *?*-file are explained in the *Source Code Control System User Guide*. The naming convention for these files is also described there.

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the “gotten” file and the *g-file*.

WARNINGS

Lines beginning with an **SOH** ASCII character (binary 001) cannot be placed in the SCCS file unless the **SOH** is escaped. This character has special meaning to SCCS (see *sccsfile*(4) (5)) and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get*/*delta* sequences should be used.

If the standard input (**−**) is specified on the *delta* command line, the **−m** (if necessary) and **−y** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

SEE ALSO

admin(1), bdiff(1), cdc(1), get(1), help(1), prs(1), rmdel(1).
sccsfile(4) in the *UNIX System V Programmer Reference Manual*.

Source Code Control System User Guide in the *UNIX System V User Guide*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

diff − differential file comparator

SYNOPSIS

diff [**−efbh**] file1 file2

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is −, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

The **−b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **−e** option produces a script of *a, c*, and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **−f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **−e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed − $1
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option **−h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **−e** and **−f** are unavailable with **−h**.

FILES

/tmp/d?????
/usr/lib/diffh for **−h**

SEE ALSO

cmp(1), comm(1), ed(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS

Editing scripts produced under the **−e** or **−f** option are naive about creating lines consisting of a single period (.).

WARNINGS

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

NAME

diff3 − 3-way differential file comparison

SYNOPSIS

diff3 [**−ex3**] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	*file1* is different
====2	*file2* is different
====3	*file3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f : *n1* **a**	Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.
f : *n1* , *n2* **c**	Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n*2, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the **−e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged ==== and ====3. Option **−x** (**−3**) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed − file1
```

FILES

/tmp/d3∗
/usr/lib/diff3prog

SEE ALSO

diff(1).

BUGS

Text lines that consist of a single . will defeat **−e**.
Files longer than 64K bytes will not work.

NAME

diffmk – mark differences between files

SYNOPSIS

diffmk name1 name2 name3

DESCRIPTION

Diffmk compares two versions of a file and creates a third file that includes "change mark" commands for *nroff* or *troff*(1). *Name1* and *name2* are the old and new versions of the file. *Diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (**.mc**) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

If anyone is so inclined, *diffmk* can be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file **macs** contains:

```
.pl 1
.ll 77
.nf
.eo
.nc `
```

The **.ll** request might specify a different line length, depending on the nature of the program being printed. The **.eo** and **.nc** requests are probably needed only for C programs.

If the characters | and * are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shell procedure).

SEE ALSO

diff(1), nroff(1), troff(1).

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

NAME

dircmp − directory comparison

SYNOPSIS

dircmp [**−d**] [**−s**] [**−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 file names common to both directories have the same contents.

−d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).

−s Suppress messages about identical files.

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

SEE ALSO

cmp(1), diff(1).

NAME

du − summarize disk usage

SYNOPSIS

du [**−ars**] [names]

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, **.** is used.

The optional argument **−s** causes only the grand total (for each of the specified *names*) to be given. The optional argument **−a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

Du is normally silent about directories that cannot be read, files that cannot be opened, etc. The **−r** option will cause *du* to generate messages in such instances.

A file with two or more links is only counted once.

BUGS

If the **−a** option is not used, non-directories given as arguments are not listed.
If there are too many distinct linked files, *du* will count the excess files more than once.
Files with holes in them will get an incorrect block count.

NAME

dump − dump selected parts of an object file

SYNOPSIS

dump [**−acfghlorst**] [**−z** name] files

DESCRIPTION

The *dump* command dumps selected parts of each of its object *file* arguments.

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:

−a	Dump the archive header of each member of each archive file argument.
−g	Dump the global symbols in the symbol table of an archive.
−f	Dump each file header.
−o	Dump each optional header.
−h	Dump section headers.
−s	Dump section contents.
−r	Dump relocation information.
−l	Dump line number information.
−t	Dump symbol table entries.
−z name	Dump line number entries for the named function.
−c	Dump the string table.

The following *modifiers* are used in conjunction with the options listed above to modify their capabilities.

−d number	Dump the section number or range of sections starting at *number* and ending either at the last section number or *number* specified by **+d**.
+d number	Dump sections in the range either beginning with first section or beginning with section specified by **−d**.
−n name	Dump information pertaining only to the named entity. This *modifier* applies to **−h**, **−s**, **−r**, **−l**, and **−t**.
−p	Supress printing of the headers.
−t index	Dump only the indexed symbol table entry. The **−t** used in conjunction with **+t**, specifies a range of symbol table entries.
+t index	Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **−t** option.
−u	Underline the name of the file for emphasis.
−v	Dump information in symbolic representation rather than numeric (e.g., C_STATIC instead of **0X02**). This *modifier* can be used with all the above options except **−s** and **−o** options of *dump*.
−z name,number	Dump line number entry or range of line numbers starting at *number* for the named function.
+z number	Dump line numbers starting at either function *name* or *number* specified by **−z,** up to *number* specified by **+z**.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the **−z** option may be replaced by a blank.

The *dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

SEE ALSO

a.out(4), ar(4) in the *UNIX System V Programmer Reference Manual.*

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).

NAME

ed, red — text editor

SYNOPSIS

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

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

DESCRIPTION

Ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. The optional **–** suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a !*shell command*. The **–p** option allows the user to specify a prompt string. If **–x** is present, an *x* command is simulated first to handle an encrypted file. *Ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via !*shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty –tabs** or **stty tab3** mode (see *stty*(1), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (**.**) alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character REs* match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. **.**, *****, **[**, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets (**[ ]**; see 1.4 below).

b. **^** (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets (**[]**) (see 1.4 below).

c. **$** (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)

1.3 A period (**.**) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets (**[]**) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (**^**), the one-character RE matches any character *except* new-line and the remaining characters in the string. The **^** has this special meaning *only* if it occurs first in the string. The minus (**−**) may be used to indicate a range of consecutive ASCII characters; for example, **[0−9]** is equivalent to **[0123456789]**. The **−** loses this special meaning if it occurs first (after an initial **^**, if any) or last in the string. The right square bracket (**]**) does not terminate such a string when it is the first character within it (after an initial **^**, if any); e.g., **[]a−f]** matches either a right square bracket (**]**) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *RE*s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*****) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{*m*\}, \{*m,*\}, or \{*m,n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m,*\} matches *at least m* occurrences; \{*m,n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression **^\(.*\)\1$** matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

3.1 A circumflex (**^**) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2 A 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 the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character **.** addresses the current line.
2. The character **$** addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. *'x* addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (*/*) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (**?**) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (**+**) or a minus sign (**−**) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with **+** or **−**, the addition or subtraction is taken with respect to the current line; e.g, **−5** is understood to mean **.−5**.
9. If an address ends with **+** or **−**, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address **−** refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character **^** in addresses is entirely equivalent to **−**.) Moreover, trailing **+** and **−** characters have a cumulative effect, so **−−** refers to the current line less 2.
10. For convenience, a comma (**,**) stands for the address pair **1,$**, while a semicolon (**;**) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n*, and *p* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; **.** is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; **.** is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; **.** is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* 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, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with **.** initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted. The **.** terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

(1,$)G/*RE***/**

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, **.** is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an **&** causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *h*elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H

The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent **?** diagnostics. It will also explain the previous **?** if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; **.** is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j

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

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address ′*x* then addresses this line; **.** is unchanged.

(.,.)l

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

(.,.)m*a*

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; **.** is left at the last line moved.

(.,.)n

The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; **.** is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p

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

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

Q

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

($)r *file*

The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; **.** is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE***/***replacement***/** or
(.,.)s/*RE***/***replacement***/g** or
(.,.)s/*RE***/***replacement***/n** n = 1-512

The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number n appears after the command, only the n th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (**&**) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of **&** in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u

The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,$)v/*RE***/***command list*

This command is the same as the global command *g* except that the *command list* is executed with . initially set to every line that does *not* match the RE.

(1,$)V/*RE***/**

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1,$)w *file*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

X

A key string is demanded from the standard input. Subsequent *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

($)=
The line number of the addressed line is typed; **.** is unchanged by this command.

!*shell command*
The remainder of the line after the ! is sent to the UNIX system shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; **.** is unchanged.

(.+1)<new-line>
An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a **?** and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

FILES

/tmp/e#	temporary; # is the process number.
ed.hup	work is saved here if the terminal is hung up.

DIAGNOSTICS

?	for command errors.
?*file*	for an inaccessible file.
	(use the *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands. It prints **?** and allows one to continue editing. A second *e* or *q* command at this point will take effect. The − command-line option inhibits this feature.

SEE ALSO
crypt(1), grep(1), sed(1), sh(1), stty(1).
fspec(4), regexp(5) in the *UNIX System V Programmer Reference Manual.*

UNIX System V Editing Guide.

CAVEATS AND BUGS

A *!* command cannot be subject to a *g* or a *v* command.
The *!* command and the **!** escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).
The sequence **\n** in a RE does not match a new-line character.
The *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 coming from a command file (i.e., ed file < ed-cmd-file), the editor will exit at the first failure of a command that is in the command file.

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.* If you are using a CRT terminal you may want to learn about the display editor *vi.*

BRIEF INTRODUCTION

To edit the contents of an existing file you begin with the command "edit name" to the shell. *Edit* makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but do not worry.

Edit prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit's* buffer (its name for the copy of the file you are editing). Most commands to *edit* use its "current line" if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and hit carriage return (as you should after all *edit* commands) this current line will be printed. If you **delete** (**d**) the current line, *edit* will print the new current line. When you start editing, *edit* makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete,** the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**) *edit* will read lines from your terminal until you give a line consisting of just a ".", placing these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append** but places the lines you give before, rather than after, the current line.

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

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

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say "[Modified]" if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (**w**) command. You can then leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will be warned that there has been "No **write** since last change" and *edit* will await another command. If you wish not to **write** the buffer out then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

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

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

The **undo** (**u**) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command which does not do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. *Edit* will give you a warning message when commands you do affect 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.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit ^D (control key and, while it is held down D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command "z.". The current line will then be the last line printed; you can get back to the line where you were before the "z." command by saying "′′". The **z** command can also be given other following characters "z−" prints a screen of text (or 24 lines) ending where you are; "z+" prints the next screenful. If you want less than a screenful of lines, type in "z.12" to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command "delete 5".

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

The current line has a symbolic name "."; this is most useful in a range of lines as in ".,$print" which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "$". Thus the command "$ delete" or "$d" deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "$−5" is the fifth before the last, and ".+20" is 20 lines after the present.

You can find out which line you are at by doing ".=". This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say "10,20delete a" which deletes these lines from the file and places them in a buffer named *a*. *Edit* has 26 such buffers named *a* through *z*. You can later get these lines back by doing "put a" to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (**e**) command after copying the lines, following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing *delete* to *yank* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move $" for example. It is not necessary to use named buffers in this case (but you can if you wish).

SEE ALSO

ex(1), vi(1).

NAME

efl − Extended Fortran Language

SYNOPSIS

efl [options] [files]

DESCRIPTION

Efl compiles a program written in the EFL language into clean Fortran on the standard output. *Efl* provides the C-like control constructs of *ratfor*(1):

statement grouping with braces.

decision-making:
if, **if-else**, and **select-case** (also known as **switch-case**);
while, **for**, Fortran **do**, **repeat**, and **repeat** ... **until** loops;
multi-level **break** and **next**.

EFL has C-like data structures, e.g.:

```
struct
        {
        integer flags(3)
        character(8) name
        long real coords(2)
        } table(100)
```

The language offers generic functions, assignment operators (**+=**, **&=**, etc.), and sequentially evaluated logical operators (**&&** and **||**). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) })
```

EFL also provides some syntactic "sugar":

free-form input:
multiple statements per line; automatic continuation; statement label names (not just numbers).

comments:
# this is a comment.

translation of relational and logical operators:
>, **>=**, **&**, etc., become **.GT.**, **.GE.**, **.AND.**, etc.

return expression to caller from function:
return (*expression*)

defines:
define *name replacement*

includes:
include *file*

Efl understands several option arguments: **−w** suppresses warning messages, **−#** suppresses comments in the generated program, and the default option **−C** causes comments to be included in the generated program.

An argument with an embedded **=** (equal sign) sets an EFL option as if it had appeared in an **option** statement at the start of the program. Many options are described in the reference manual. A set of defaults for a particular target machine may be selected by one of the choices: **system=unix**, **system=gcos**, or **system=cray**. The default setting of the **system** option is the same as the machine the compiler is running on.

Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.

Efl is best used with *f77*(1).

SEE ALSO

cc(1), f77(1), ratfor(1).

NAME

enable, disable — enable/disable LP printers

SYNOPSIS

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

DESCRIPTION

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

Disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

−c	Cancel any requests that are currently printing on any of the designated printers.
−r[*reason*]	Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **−r** option. If the **−r** option is not present or the **−r** option is given without a reason, then a default reason will be used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).

NAME

env — set environment for command execution

SYNOPSIS

env [—] [name=value] ... [command args]

DESCRIPTION

Env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name*=*value* are merged into the inherited environment before the command is executed. The — flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO

sh(1).
exec(2), profile(4), environ(5) in the *UNIX System V Programmer Reference Manual.*

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*.

DOCUMENTATION

The *Ex Reference Manual* is a comprehensive and complete manual for the command mode features of *ex*, but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex* see the editing documents written by Brian Kernighan for the editor *ed;* the material in the introductory and advanced documents works also with *ex*.

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi*. The *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction*. The *vi*(1) manual page can also be used as reference.

FOR ED USERS

If you have used *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**) and which is the central mode of editing when using *vi*(1).

Ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Hitting ^D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting 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 clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the telephone, you can use the editor **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

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

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 new 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*.

INVOCATION OPTIONS

The following invocation options are interpreted by *ex*:

–	Suppress all interactive-user feedback. This is useful in processing editor scripts.
–v	Invokes *vi*
–t *tagfR*	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 will be printed.
–R	*Readonly* mode set, prevents accidentally overwriting the file.
+*command*	Begin editing by executing the specified editor search or positioning *command*.
–l	**LISP** mode; indents appropriately for lisp code, the **()** **{}** **[[** and **]]** commands in *vi* are modified to have meaning for *lisp*.
–x	Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.

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

AUTHOR

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

FILES

/usr/lib/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) in the *UNIX System V Programmer Reference Manual.*

CAVEATS AND BUGS

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.

NAME

expr − evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within {} symbols.

expr \| *expr*
: returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr \& *expr*
: returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { =, \>, \>=, \<, \<=, != } *expr*
: returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, − } *expr*
: addition or subtraction of integer-valued arguments.

expr { *, /, % } *expr*
: multiplication, division, or remainder of the integer-valued arguments.

expr : *expr*
: The matching operator : compares the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of *ed*(1), except that all patterns are "anchored" (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

EXAMPLES

1.
```
a=`expr $a + 1`
```
adds 1 to the shell variable **a**.

2.
```
# 'For $a equal to either "/usr/abc/file" or just "file"'
expr $a : '.*/\(.*\)' \| $a
```
returns the last segment of a path name (i.e., file). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. # A better representation of example 2.

```
expr //$a : '.*/\(.*\)'
```

The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4.

```
expr $VAR : '.*'
```

returns the number of characters in **$VAR**.

SEE ALSO

ed(1), sh(1).

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

BUGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an **=**, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the **=** operator). The following works:

```
expr X$a = X=
```

NAME

f77 − Fortran 77 compiler

SYNOPSIS

f77 [options] files

DESCRIPTION

F77 is the UNIX System Fortran 77 compiler; it accepts several types of *file* arguments:

Arguments whose names end with **.f** are taken to be Fortran 77 source programs; they are compiled and each object program is left in the current directory in a file whose name is that of the source, with **.o** substituted for **.f**.

Arguments whose names end with **.r** or **.e** are taken to be RATFOR or EFL source programs, respectively. These are first transformed by the appropriate preprocessor, then compiled by *f77*, producing **.o** files.

In the same way, arguments whose names end with **.c** or **.s** are taken to be C or assembly source programs and are compiled or assembled, producing **.o** files.

The following *options* have the same meaning as in *cc*(1) (see *ld*(1) for link editor options):

−c	Suppress link editing and produce **.o** files for each source file.
−p	Prepare object files for profiling (see *prof*(1)).
−O	Invoke an object-code optimizer.
−S	Compile the named programs and leave the assembler-language output in corresponding files whose names are suffixed with **.s**. (No **.o** files are created.)
−o*output*	Name the final output file *output*, instead of **a.out**.
−f	In systems without floating-point hardware, use a version of *f77* that handles floating-point constants and links the object program with the floating-point interpreter.
−g	Generate additional information needed for the use of *sdb*(1).

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	Do not "fold" cases. *F77* is normally a no-case language (i.e., **a** is equal to **A**). The **−U** option causes *f77* to treat upper and lower cases to be separate.
−u	Make the default type of a variable *undefined*, rather than using the default Fortran rules.
−v	*Verbose* mode. Provide diagnostics for each process during compilation.
−w	Suppress all warning messages. If the option is **−w66**, only Fortran 66 compatibility warnings are suppressed.
−F	Apply EFL and RATFOR preprocessor to relevant files, put the result in files whose names have their suffix changed to **.f**. (No **.o** files are created.)
−m	Apply the M4 preprocessor to each EFL or RATFOR source file before transforming with the *ratfor*(1) or *efl*(1) processors.
−E	The remaining characters in the argument are used as an EFL flag argument whenever processing a **.e** file.

−R The remaining characters in the argument are used as a RATFOR flag argument whenever processing a **.r** file.

Other arguments are taken to be either link-editor option arguments or *f77*-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	pass 2
/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 Section 3 of this Manual.

SEE ALSO

asa(1), cc(1), efl(1), fsplit(1), ld(1), m4(1), prof(1), ratfor(1), sdb(1).

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the link editor *ld*(1).

NAME

factor − factor a number

SYNOPSIS

factor [number]

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{56} (about 7.2×10^{16}) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to $\sqrt{n}$ and occurs when n is prime or the square of a prime. It takes 1 minute to factor a prime near 10^{14} on a PDP-11.

DIAGNOSTICS

"Ouch" for input out of range or for garbage input.

NAME

file − determine file type

SYNOPSIS

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

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable **a.out**, *file* will print the version stamp, provided it is greater than 0 (see *ld*(1)).

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

File uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of **/etc/magic** explains its format.

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

The **−c** flag causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **−c**.

SEE ALSO

ld(1).

NAME

find − find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where +*n* means more than *n*, −*n* means less than *n* and *n* means exactly *n*.

−name *file*	True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for **[**, **?** and *).
−perm *onum*	True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared.
−type *c*	True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **p**, or **f** for block special file, character special file, directory, fifo (a.k.a named pipe), or plain file respectively.
−links *n*	True if the file has *n* links.
−user *uname*	True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.
−group *gname*	True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.
−size *n*[**c**]	True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.
−atime *n*	True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find* itself.
−mtime *n*	True if the file has been modified in *n* days.
−ctime *n*	True if the file has been changed in *n* days.
−exec *cmd*	True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current path name.
−ok *cmd*	Like **−exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.
−print	Always true; causes the current path name to be printed.
−cpio *device*	Always true; write the current file on *device* in *cpio*(4) format (5120-byte records).
−newer *file*	True if the current file has been modified more recently than the argument *file*.

−depth Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files that are contained in directories without write permission.

(*expression*) True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

1) The negation of a primary (**!** is the unary *not* operator).

2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

3) Alternation of primaries (**−o** is the *or* operator).

EXAMPLE

To remove all files named **a.out** or ***.o** that have not been accessed for a week:

```
find  /  \( −name a.out −o −name '*.o' \) −atime +7 −exec rm {} \;
```

FILES

/etc/passwd, /etc/group

SEE ALSO

chmod(1), cpio(1), sh(1), test(1).
stat(2), cpio(4), fs(4) in the *UNIX System V Programmer Reference Manual.*

NAME

fsplit – split f77, ratfor, or efl files

SYNOPSIS

fsplit options files

DESCRIPTION

Fsplit splits the named *file(s)* into separate files, with one procedure per file. A procedure includes *blockdata*, *function*, *main*, *program*, and *subroutine* program segments. Procedure *X* is put in file *X*.**f**, *X*.**r**, or *X*.**e** depending on the language option chosen, with the following exceptions: *main* is put in the file *MAIN*.**[efr]** and unnamed *blockdata* segments in the files *blockdataN*.**[efr]** where *N* is a unique integer value for each file.

The following *options* pertain:

−f	(default) Input files are *f77*.
−r	Input files are *ratfor*.
−e	Input files are *Efl*.
−s	Strip *f77* input lines to 72 or fewer characters with trailing blanks removed.

SEE ALSO

csplit(1), efl(1), f77(1), ratfor(1), split(1).

NAME

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 viewport's lower left corner to *n* inches.
xv*n*	Set width of viewport to *n* inches.
yd*n*	Set y displacement of the viewport's lower left corner 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) in the *UNIX System V Programmer Reference Manual.*

NAME

ged − graphical editor

SYNOPSIS

ged [**−euRr***n*] [GPS file **...**]

DESCRIPTION

Ged is an interactive graphical editor used to display, construct, and edit GPS files on TEKTRONIX 4010 series display terminals. If GPS *file*(s) are given, *ged* reads them into an internal display buffer and displays the buffer. The GPS in the buffer can then be edited. If − is given as a file name, *ged* reads a GPS from the standard input.

Ged accepts the following command line options:

e Do not erase the screen before the initial display.

r*n* Display region number *n*.

u Display the entire GPS *universe*.

R Restricted shell invoked on use of **!**.

A GPS file is composed of instances of three graphical objects: *lines*, *arc*, and *text*. *Arc* and *lines* objects have a start point, or *object-handle*, followed by zero or more points, or *point-handles*. *Text* has only an object-handle. The objects are positioned within a Cartesian plane, or *universe*, having 64K (−32K to +32K) points, or *universe-units*, on each axis. The universe is divided into 25 equal sized areas called *regions*. Regions are arranged in five rows of five squares each, numbered 1 to 25 from the lower left of the universe to the upper right.

Ged maps rectangular areas, called *windows*, from the universe onto the display screen. Windows allow the user to view pictures from different locations and at different magnifications. The *universe-window* is the window with minimum magnification, i.e., the window that views the entire universe. The *home-window* is the window that completely displays the contents of the display buffer.

COMMANDS

Ged commands are entered in *stages*. Typically each stage ends with a **<cr>** (return). Prior to the final **<cr>** the command may be aborted by typing **rubout**. The input of a stage may be edited during the stage using the erase and kill characters of the calling shell. The prompt * indicates that *ged* is waiting at stage 1.

Each command consists of a subset of the following stages:

1. *Command line*
   A *command line* consists of a command *name* followed by *argument*(s) followed by a **<cr>**. A command *name* is a single character. Command *arguments* are either *option*(s) or a *file-name*. *Options* are indicated by a leading −.

2. *Text* *Text* is a sequence of characters terminated by an unescaped **<cr>** (120 lines of text maximum).

3. *Points* *Points* is a sequence of one or more screen locations (maximum of 30) indicated either by the terminal crosshairs or by name. The prompt for entering *points* is the appearance of the crosshairs. When the crosshairs are visible, typing:

   sp (space) enters the current location as a *point*. The *point* is identified with a number.

$*n*	enters the previous *point* numbered *n*.
>*x*	labels the last *point* entered with the upper case letter *x*.
$*x*	enters the *point* labeled *x*.
.	establishes the previous *points* as the current *points*. At the start of a command the previous *points* are those locations given with the previous command.
=	echoes the current *points*.
$.*n*	enters the *point* numbered *n* from the previous *points*.
#	erases the last *point* entered.
@	erases all of the *points* entered.

4. *Pivot* The *pivot* is a single location, entered by typing **<cr>** or by using the **$** operator, and indicated with a *.

5. *Destination*
The *destination* is a single location entered by typing **<cr>** or by using **$**.

COMMAND SUMMARY

In the summary, characters typed by the user are printed in **bold**. Command stages are printed in *italics*. Arguments surrounded by brackets "[]" are optional. Parentheses "()" surrounding arguments separated by "or" means that exactly one of the arguments must be given.

Construct commands:

Arc	[−**e**cho,**s**tyle,**w**eight] *points*
Box	[−**e**cho,**s**tyle,**w**eight] *points*
Circle	[−**e**cho,**s**tyle,**w**eight] *points*
Hardware	[−**e**cho] *text points*
Lines	[−**e**cho,**s**tyle,**w**eight] *points*
Text	[−**a**ngle,**e**cho,**h**eight,**m**id-point,**r**ight-point,**t**ext,**w**eight] *text points*

Edit commands:

Delete	(− (**u**niverse or **v**iew) or *points*)
Edit	[−**a**ngle,**e**cho,**h**eight,**s**tyle,**w**eight] (− (**u**niverse or **v**iew) or *points*)
Kopy	[−**e**cho,**p**oints,**x**] *points pivot destination*
Move	[−**e**cho,**p**oints,**x**] *points pivot destination*
Rotate	[−**a**ngle,**e**cho,**k**opy,**x**] *points pivot destination*
Scale	[−**e**cho,**f**actor,**k**opy,**x**] *points pivot destination*

View commands:

coordinates	*points*
erase	
new-display	
object-handles	(− (**u**niverse or **v**iew) or *points*)

point-handles	(− (**l**abelled-points or **u**niverse or **v**iew) or *points*)
view	(− (**h**ome or **u**niverse or **r**egion) or [−**x**] *pivot destination*)
x	[−view] *points*
zoom	[−**out**] *points*

Other commands:

quit or **Q**uit	
read	[−**a**ngle,**e**cho,**h**eight,**m**id-point,**r**ight-point,**t**ext,**w**eight *file-name* [*destination*]
set	[−**a**ngle,**e**cho,**f**actor,**h**eight,**k**opy,**m**id-point,**p**oints, **r**ight-point,**s**tyle,**t**ext,**w**eight,**x**]
write	*file-name*
!*command*	
?	

Options:

Options specify parameters used to construct, edit, and view graphical objects. If a parameter used by a command is not specifed as an *option*, the default value for the parameter will be used (see set below). The format of command *options* is:

−*option*[,*option*]

where *option* is *keyletter*[*value*]. Flags take on the *values* of true or false indicated by + and − respectively. If no *value* is given with a flag, true is assumed.

Object options:

angle*n*	Angle of *n* degrees.
echo	When true, echo additions to the display buffer.
factor*n*	Scale factor is *n* percent.
height*n*	Height of *text* is *n* universe-units ($0 \leqslant n < 1280$).
kopy	When true, copy rather than move.
mid-point	When true, mid-point is used to locate text string.
points	When true, operate on points; otherwise operate on objects.
right-point	When true, right-point is used to locate *text* string.
style*type*	Line style set to one of following *types*:

so	solid
da	dashed
dd	dot-dashed
do	dotted
ld	long-dashed

text	When false, *text* strings are outlined rather than drawn.
weight*type*	Sets 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 **L**ines

behave similarly. Each consists of a *command line* followed by *points*. The first *point* entered is the object-handle. Successive *points* are point-handles. **L**ines connect the handles in numerical order. **A**rc fits a curve to the handles (currently a maximum of 3 points will be fit with a circular arc; splines will be added in a later version).

Box and **C**ircle

are special cases of **L**ines and **A**rc, respectively. **B**ox generates a rectangle with sides parallel to the universe axes. A diagonal of the rectangle would connect the first *point* entered with the last *point*. The first *point* is the object-handle. Point-handles are created at each of the vertices. **C**ircle 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 **H**ardware

generate *text* objects. Each consists of a *command line*, *text* and *points*. *Text* is a sequence of characters delimited by **<cr>**. Multiple lines of text may be entered by preceding a **cr** with a backslash (i.e., **\cr**). The **T**ext 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 **H**ardware command sends the characters in *text* uninterpreted to the terminal.

Edit commands:

Edit commands operate on portions of the display buffer called *defined areas*. A defined area is referenced either with an area *option* or interactively. If an area *option* is not given, the perimeter of the defined area is indicated by *points*. If no *point* is entered, a small defined area is built around the location of the **<cr>**. This is useful to reference a single *point*. If only one *point* is entered, the location of the **<cr>** is taken in conjunction with the *point* to indicate a diagonal of a rectangle. A defined area referenced by *points* will be outlined with dotted lines.

Delete

removes all objects whose object-handle lies within a defined area. The **u**niverse 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 **M**ove)
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 **k**opy flag is true then the objects are copied rather than moved.

Scale
For objects whose object handles are within a defined area, point displacements from the *pivot* are scaled by **f**actor percent. If the **k**opy 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 **p**oint-handles)
labels object-handles (and/or point-handles) that lie within the defined area with **O** (or **P**). **P**oint-handles identifies labeled points when the **l**abelled-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 **h**ome, **u**niverse, and **r**egion display particular windows in the universe.

x indicates the center of a defined area. Option **v**iew indicates the center of the screen.

zoom
decreases (**z**oom **o**ut) 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 **Q**uit
exit from *ged.* **Q**uit 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) in the *UNIX System V Programmer Manual.*

An Introduction to the Graphical Editor in the *UNIX System V Graphics Guide*.

WARNING

See Appendix A of the *TEKTRONIX 4014 Computer Display Terminal User's Manual* for the proper terminal strap options.

NAME

get − get a version of an SCCS file

SYNOPSIS

get [**−r**SID] [**−c**cutoff] [**−i**list] [**−x**list] [**−w**string] [**−a**seq-no.] [**−k**] [**−e**] [**−l**[p]] [**−p**] [**−m**] [**−n**] [**−s**] [**−b**] [**−g**] [**−t**] file ...

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with **−**. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of **−** is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

−r*SID* 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** keyletter is also used), as a function of the SID specified.

−c*cutoff* *Cutoff* date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **−c7502** is equivalent to **−c750228235959**. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: "**−c77/2/2 9:22:25**". Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

```
~!get "−c%E% %U%" s.file
```

−e Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The **−e** keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *get*s for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **get −e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an **−e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the **−k** keyletter in place of the **−e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **−e** keyletter is used.

−b Used with the **−e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

−i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID − SID
```

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

−x*list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **−i** keyletter for the *list* format.

−k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **−k** keyletter is implied by the **−e** keyletter.

−l[p] Causes a delta summary to be written into an *l-file*. If **−lp** is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.

−p Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **−s** keyletter is used, in which case it disappears.

−s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

−m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

−n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **−m** and **−n** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **−m** keyletter generated format.

−g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

−t Used to access the most recently created ("top") delta in a given release (e.g., **−r1**), or release and level (e.g., **−r1.2**).

−w *string* Substitute *string* for all occurrences of **@(#)get.1 6.2** when *get*ing the file.

−**a***seq-no.* The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(5)). This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter, and users should not use it. If both the **−r** and **−a** keyletters are specified, the **−a** keyletter is used. Care should be taken when using the **−a** keyletter in conjunction with the **−e** keyletter, as the SID of the delta to be created may not be what one expects. The **−r** keyletter can be used with the **−a** and **−e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **−e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **−i** keyletter is used included deltas are listed following the notation "Included"; if the **−x** keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	**−b** Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	−	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	−	Trunk succ.# 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

* "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.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

# Successor.

† The **−b** keyletter is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of − means "irrelevant".

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

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 keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. 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 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, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **−p** keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **−k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **−l** keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
 * otherwise.
b. A blank character if the delta was applied or 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** keyletter).
d. Blank.
e. SCCS identification (SID).
f. Tab character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. Login name of person who created *delta*.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an **−e** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **−e** keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **−i** keyletter argument if it was present, followed by a blank and the **−x** keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1).
sccsfile(4) in the *UNIX System V Programmer Reference Manual.*

Source Code Control System in the *UNIX System V Support Tools Guide.*

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the **−e** keyletter is used.

NAME

getopt − parse command options

SYNOPSIS

set −− `getopt** optstring **$*`

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *Optstring* is a string of recognized option letters (see *getopt*(3C)); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option −− is used to delimit the end of the options. If it is used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it; in either case, *getopt* will place it at the end of the options. The positional parameters ($1 $2 ...) of the shell are reset so that each option is preceded by a − and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b)        FLAG=$i; shift;;
        -o)             OARG=$2; shift 2;;
        --)             shift; break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

sh(1), getopt(3C).

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

graph − draw a graph

SYNOPSIS

graph [options]

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *tplot*(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes ", in which case they may be empty or contain blanks and numbers; labels never contain new-lines.

The following options are recognized, each as a separate argument:

−a	Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **−x**).
−b	Break (disconnect) the graph after each label in the input.
−c	Character string given by next argument is default label for each point.
−g	Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).
−l	Next argument is label for graph.
−m	Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (e.g., the TEKTRONIX 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).
−s	Save screen, do not erase before plotting.
−x [l]	If **l** is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.
−y [l]	Similarly for *y*.
−h	Next argument is fraction of space for height.
−w	Similarly for width.
−r	Next argument is fraction of space to move right before plotting.
−u	Similarly to move up before plotting.
−t	Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **−s** option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

graphics(1G), spline(1G), tplot(1G).

BUGS

Graph stores all points internally and drops those for which there is no room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

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, type **EOT** (control-d on most terminals). To logoff from the graphics environment, type **quit**.

The command line format for a command in *graphics* is *command name* followed by *argument*(s). An *argument* may be a *file name* or an *option string.* A *file name* is the name of any UNIX 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 have been partitioned into four groups.

Commands that manipulate and plot numerical data; see *stat*(1G).

Commands that generate tables of contents; see *toc*(1G).

Commands that interact with graphical devices; see *gdev*(1G) and *ged*(1G).

A collection of graphical utility commands; see *gutil*(1G).

A list of the *graphics* commands can be generated by typing **whatis** in the *graphics* environment.

SEE ALSO

gdev(1G), ged(1G), gutil(1G), stat(1G), toc(1G).
gps(4) in the *UNIX System V Programmer Reference Manual.*

UNIX System V Graphics Guide.

NAME

greek – select terminal filter

SYNOPSIS

greek [–Tterminal]

DESCRIPTION

Greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character TELETYPE Model 37 terminal (which is the *nroff*(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 *terminals* are recognized currently:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	TEKTRONIX 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	TEKTRONIX 4014.

FILES

/usr/bin/300
/usr/bin/300s
/usr/bin/4014
/usr/bin/450
/usr/bin/hp

SEE ALSO

300(1), 4014(1), 450(1), eqn(1), hp(1), mm(1), nroff(1), tplot(1G).
environ(5), greek(5), term(5) in the *UNIX System V Programmer Reference Manual.*

NAME

grep, egrep, fgrep – search a file for a pattern

SYNOPSIS

grep [options] expression [files]

egrep [options] [expression] [files]

fgrep [options] [strings] [files]

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular *expressions* in the style of *ed*(1); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

–v All lines but those matching are printed.
–x (Exact) only lines matched in their entirety are printed (*fgrep* only).
–c Only a count of matching lines is printed.
–i Ignore upper/lower case distinction during comparisons.
–l Only the names of files with matching lines are listed (once), separated by new-lines.
–n Each line is preceded by its relative line number in the file.
–b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.
–s The error messages produced for nonexistent or unreadable files are suppressed (*grep* only).
–e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a – (does not work with *grep*).
–f *file*
The regular *expression* (*egrep*) or *strings* list (*fgrep*) is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters **$**, *****, **[**, **^**, **|**, **(**, **)**, and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'.

Fgrep searches for lines that contain one of the *strings* separated by new-lines.

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

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

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

SEE ALSO

ed(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *grep*, but we do not know a single algorithm that spans a wide enough range of space-time tradeoffs.
Lines are limited to BUFSIZ characters; longer lines are truncated. (BUFSIZ is defined in **/usr/include/stdio.h.**)
Egrep does not recognize ranges, such as **[a −z]**, in character classes.
If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

NAME

gutil − graphical utilities

SYNOPSIS

command-name [options] [files]

DESCRIPTION

Below is a list of miscellaneous device independent utility commands found in **/usr/bin/graf**. If no *files* are given, input is from the standard input. All output is to the standard output. Graphical data is stored in GPS format; see *gps*(4).

bel − send bel character to terminal

cvrtopt [=**s***string* **f***string* **i***string* **t***string*] [*args*] − options converter
Cvrtopt reformats *args* (usually the command line arguments of a calling shell procedure) to facilitate processing by shell procedures. An *arg* is either a file name (a string not beginning with a −, or a − by itself) or an option string (a string of options beginning with a −). Output is of the form:

−option −option . . . file name(s)

All options appear singularly and preceding any file names. Options that take values (e.g., −r1.1) or are two letters long must be described through options to *cvrtopt*.

Cvrtopt is usually used with *set* in the following manner as the first line of a shell procedure:

set − `cvrtopt =[***options***] $@`

Options to *cvrtopt* are:

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.

String is a one- or two-letter option name.

gd [GPS *files*] − GPS dump
Gd prints a human readable listing of GPS.

gtop [−**r***n***u**] [GPS *files*] − GPS to *plot*(4) filter
Gtop transforms a GPS into *plot*(4) commands displayable by *plot* filters. GPS objects are translated if they fall within the window that circumscribes the first *file* unless an *option* is given.
Options:

r*n*	translate objects in GPS **r**egion *n*.
u	translate all objects in the GPS **u**niverse.

pd [*plot*(5) *files*] − *plot*(4) dump
Pd prints a human readable listing of *plot*(4) format graphical commands.

ptog [*plot*(5) *files*] − *plot*(4) to GPS filter
Ptog transforms *plot*(4) commands into a GPS.

quit − terminate session

remcom [*files*] − remove comments
Remcom copies its input to its output with comments removed. Comments are as defined in C (i.e., /* comment */).

whatis [**−o**] [*names*] − brief on-line documentation
Whatis prints a brief description of each *name* given. If no *name* is given, then the current list of description *names* is printed. The command **whatis** \* prints out every description.
Option:

o	just print command **o**ptions

yoo *file* − pipe fitting
Yoo is a piping primitive that deposits the output of a pipeline into a *file* used in the pipeline. Note that, without *yoo*, this is not usually successful as it causes a read and write on the same file simultaneously.

SEE ALSO

graphics(1G).
gps(4), plot(4) in the *UNIX System V Programmer Reference Manual.*

NAME

help − ask for help

SYNOPSIS

help [args]

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1	Begins with non-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**)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

FILES

/usr/lib/help	directory containing files of message text.
/usr/lib/help/helploc	file containing locations of help files not in **/usr/lib/help**.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

hp − handle special functions of Hewlett-Packard 2640 and 2621-series terminals

SYNOPSIS

hp [−e] [−m]

DESCRIPTION

Hp supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most *nroff* output. A typical use is:

```
nroff −h files ... | hp
```

Regardless of the hardware options on your terminal, *hp* tries to do sensible things with underlining and reverse line-feeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

The flags are as follows:

−e It is assumed that your terminal has the "display enhancements" feature, and so maximal use is made of the added display modes. Overstruck characters are presented in the Underline mode. Superscripts are shown in Half-bright mode, and subscripts in Half-bright, Underlined mode. If this flag is omitted, *hp* assumes that your terminal lacks the "display enhancements" feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode, i.e., dark-on-light, rather than the usual light-on-dark.

−m Requests minimization of output by removal of new-lines. Any contiguous sequence of 3 or more new-lines is converted into a sequence of only 2 new-lines; i.e., any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, *hp* provides the same set as does *300*(1), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown. The display is adequate for examining output from *neqn*.

DIAGNOSTICS

"line too long" if the representation of a line exceeds 1,024 characters.
The exit codes are **0** for normal termination, **2** for all errors.

SEE ALSO

300(1), col(1), eqn(1), greek(1), nroff(1), tbl(1).

BUGS

An "overstriking sequence" is defined as a printing character followed by a backspace followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in Inverse Video; otherwise, only the first printing character is shown (again, underlined or in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse line-feeds, backspaces) can make text "disappear"; in particular, tables generated by *tbl*(1) that contain vertical lines will often be missing the lines of text that contain the "foot" of a vertical line, unless the input to *hp* is piped through *col*(1).
Although some terminals do provide numerical superscript characters, no attempt is made to display them.

NAME

hyphen − find hyphenated words

SYNOPSIS

hyphen [files]

DESCRIPTION

Hyphen finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, *hyphen* may be used as a filter.

EXAMPLE

The following will allow the proofreading of *nroff* hyphenation in *textfile*.

```
mm textfile | hyphen
```

SEE ALSO

mm(1), nroff(1).

BUGS

Hyphen cannot cope with hyphenated *italic* (i.e., underlined) words; it will often miss them completely, or mangle them.
Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

NAME

id – print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

Id writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

SEE ALSO

logname(1).
getuid(2) in the *UNIX System V Programmer Reference Manual.*

NAME

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

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

Ipcrm will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

–q *msqid*	removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.
–m *shmid*	removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
–s *semid*	removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.
–Q *msgkey*	removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.
–M *shmkey*	removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
–S *semkey*	removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl*(2), *shmctl*(2), and *semctl*(2). The identifiers and keys may be found by using *ipcs*(1).

SEE ALSO

ipcs(1).
msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2) in the *UNIX System V Programmer Reference Manual.*

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 following *options*:

380.sp0u

−q Print information about active message queues.
−m Print information about active shared memory segments.
−s Print information about active semaphores.

If any of the options **−q**, **−m**, or **−s** are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed.

−b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.

−c Print creator's login name and group name. See below.

−o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

−p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.

−t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop*(2) on semaphores.) See below.

−a Use all print *options*. (This is a shorthand notation for **−b**, **−c**, **−o**, **−p**, and **−t**.)

−C *corefile*
Use the file *corefile* in place of **/dev/kmem**.

−N *namelist*
The argument will be taken as the name of an alternate *namelist* (**/unix** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all)
Type of the facility:

q	message queue;
m	shared memory segment;
s	semaphore.

ID (all)
The identifier for the facility entry.

KEY (all)
The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

MODE (all)
The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:
The first two characters are:

R	if a process is waiting on a *msgrcv*;
S	if a process is waiting on a *msgsnd*;
D	if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
C	if the associated shared memory segment is to be cleared when the first attach is executed;
–	if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

r	if read permission is granted;
w	if write permission is granted;
a	if alter permission is granted;
–	if the indicated permission is *not* granted.

OWNER (all)
The login name of the owner of the facility entry.

GROUP (all)
The group name of the group of the owner of the facility entry.

CREATOR (a,c)
The login name of the creator of the facility entry.

CGROUP (a,c)
The group name of the group of the creator of the facility entry.

CBYTES (a,o)
The number of bytes in messages currently outstanding on the associated message queue.

QNUM (a,o)
The number of messages currently outstanding on the associated message queue.

QBYTES (a,b)
The maximum number of bytes allowed in messages outstanding on the associated message queue.

LSPID (a,p)
The process ID of the last process to send a message to the associated queue.

LRPID (a,p)
The process ID of the last process to receive a message from the associated queue.

STIME (a,t)
The time the last message was sent to the associated queue.

RTIME (a,t)
The time the last message was received from the associated queue.

CTIME (a,t)
The time when the associated entry was created or changed.

NATTCH (a,o)
The number of processes attached to the associated shared memory segment.

SEGSZ (a,b)
The size of the associated shared memory segment.

CPID (a,p)
The process ID of the creator of the shared memory entry.

LPID (a,p)
The process ID of the last process to attach or detach the shared memory segment.

ATIME (a,t)
The time the last attach was completed to the associated shared memory segment.

DTIME (a,t)
The time the last detach was completed on the associated shared memory segment.

NSEMS (a,b)
The number of semaphores in the set associated with the semaphore entry.

OTIME (a,t)
The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgop(2), semop(2), shmop(2) in the *UNIX System V Programmer Reference Manual.*

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

NAME

join − relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is −, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively. The following options are recognized:

−a*n* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.

−e *s* Replace empty output fields by string *s*.

−j*n m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with **1**.

−o *list* Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.

−t*c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
join −j1 4 −j2 3 −o 1.1 2.1 1.6 −t: /etc/passwd /etc/group
```

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

BUGS

With default field separation, the collating sequence is that of **sort −b**; with **−t**, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq* and *awk*(1) are wildly incongruous.

Filenames that are numeric may cause conflict when the **-o** option is used right before listing filenames.

NAME

kill − terminate a process

SYNOPSIS

kill [−signo] PID ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with **&** is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by − is given as first argument, that signal is sent instead of terminate (see *signal*(2)). In particular "kill −9 ..." is a sure kill.

SEE ALSO

ps(1), sh(1).
kill(2), signal(2) in the *UNIX System V Programmer Reference Manual.*

NAME

ld − link editor for common object files

SYNOPSIS

ld [options] filename

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**. By default this file is executable if no errors occurred during the load. If any input file, *file-name*, is not an object file, *ld* assumes it is either an archive library or a text file containing link editor directives. (See the *Link Editor User Guide* in the *UNIX System V Programmer Guide* for a discussion of input directives.)

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The library (archive) symbol table (see *ar*(4)) is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is unimportant.

The following options are recognized by *ld*.

−e epsym
: Set the default entry point address for the output file to be that of the symbol *epsym*.

−f fill
: Set the default fill pattern for "holes" within an output section as well as initialized *bss* sections. The argument *fill* is a two-byte constant.

−l*x*
: Search a library **lib***x***.a,** where *x* is up to seven characters. A library is searched when its name is encountered, so the placement of a **−l** is significant. By default, libraries are located in **/lib** and **/usr/lib/**.

−m
: Produce a map or listing of the input/output sections on the standard output.

−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. The link editor will not complain about unresolved references.

−s
: Strip line number entries and symbol table information from the output object file.

−t
: Turn off the warning about multiply-defined symbols that are not the same size.

−u symname
: Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

−x Do not preserve local (non-.globl) symbols in the output symbol table; enter external and static symbols only. This option saves some space in the output file.

−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.

−M Output a message for each multiply-defined external definition. However, if the objects being loaded include debugging information, extraneous output is produced (see the **−g** option in *cc*(1)).

−N Put the data section immediately following the text in the output file.

−V Output a message giving information about the version of ld being used.

−VS num
Use **num** as a decimal version stamp identifying the **a.out** file that is produced. The version stamp is stored in the optional header.

FILES

/lib/lib*x*.a	libraries
/usr/lib/lib*x*.a	libraries
a.out	output file

SEE ALSO

as(1), cc(1).
exit(2), a.out(4), ar(4) in the *UNIX System V Programmer Reference Manual.*

CAVEATS

Through its options and input directives, the common link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives and options should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the data space.
- When the link editor is called through *cc*(1), a startup routine is linked with the user's 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.

NAME

lex − generate programs for simple lexical tasks

SYNOPSIS

lex [**−rctvn**] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file **lex.yy.c** is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in **[abx−z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators *****, **+**, and **?** mean respectively any non-negative number of, any positive number of, and either zero or one occurrence of, the previous character or character class. The character **.** is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported. The notation *r*{*d*,*e*} in a rule indicates between *d* and *e* instances of regular expression *r*. It has higher precedence than |, but lower than *, ?, +, and concatenation. The character ^ at the beginning of an expression permits a successful match only immediately after a new-line, and the character **$** at the end of an expression requires a trailing new-line. The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \. Thus **[a−zA−Z]+** matches a string of letters.

Three subroutines defined as macros are expected: **input()** to read a character; **unput(***c***)** to replace a character read; and **output(***c***)** to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex()**, and the library contains a **main()** which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function **yymore()** accumulates additional characters into the same *yytext*; and the function **yyless(***p***)** pushes back the portion of the string matched beginning at *p*, which should be between *yytext* and *yytext+yyleng*. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes %% it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a %%, as in YACC. Lines preceding %% which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with **{}**. Note that curly brackets do not imply parentheses; only string substitution is done.

EXAMPLE

```
D       [0−9]
%%
if      printf("IF statement\n");
[a−z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
                        {
                        case '/': break;
                        case '*': unput('*');
                        default: go to loop;
                        }
                }
```

The external names generated by *lex* all begin with the prefix **yy** or **YY**.

The flags must appear before any files. The flag **−r** indicates RATFOR actions, **−c** indicates C actions and is the default, **−t** causes the **lex.yy.c** program to be written instead to standard output, **−v** provides a one-line summary of statistics of the machine generated, **−n** will not print out the − summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%**p** *n*	number of positions is *n* (default 2000)
%**n** *n*	number of states is *n* (500)
%**t** *n*	number of parse tree nodes is *n* (1000)
%**a** *n*	number of transitions is *n* (3000)

The use of one or more of the above automatically implies the **−v** option, unless the **−n** option is used.

SEE ALSO

yacc(1).
malloc(3X) in the *UNIX System V Programmer Reference Manual.*

BUGS

The **−r** option is not yet fully operational.

NAME
line − read one line

SYNOPSIS
line

DESCRIPTION
Line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO
sh(1).
read(2) in the *UNIX System V Programmer Reference Manual.*

NAME

lint − a C program checker

SYNOPSIS

lint [option] ... file ...

DESCRIPTION

Lint attempts to detect features of the C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that 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 none 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 will take all the **.c**,**.ln**, and **llib-l***x***.ln** (specified by **−l***x*) files and process 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.

Any number of *lint* options may be used, in any order, intermixed with file-name arguments. The following options are used to suppress certain kinds of complaints:

−a Suppress complaints about assignments of long values to variables that are not long.

−b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in many such complaints).

−h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

−u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program).

−v Suppress complaints about unused arguments in functions.

−x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint*'s behavior:

−l*x* Include additional lint library **llib-l***x***.ln**. For example, 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 reference 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" (see below).

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. The preprocessor 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 will 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 and function arguments 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 have been processed. Finally, 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 will be 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 will print 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.

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).

BUGS

exit(2), *longjmp*(3C), and other functions that do not return are not understood; this causes various lies.

NAME

ln—create a symbolic link in WorkNet

SYNOPSIS

ln —s [@system] [filename] [@system] [symbolic link name]

DESCRIPTION

Use *ln* to create a symbolic link within one system or across system boundaries to a remote system.

A symbolic link can be used to establish a link to a remote file or directory from the local system. A symbolic link does not contain any data; it simply points to the referenced file where the data resides. This allows application programs to access data on different computers.

Create symbolic links judiciously, especially if you are linking between systems. If either system is disconnected from the network, the link will be pointing to a file or directory that cannot be accessed.

EXAMPLES

In the following example, /etc/termcap is the file you are referencing, and /usr/sarah/terms is the name of the symbolic link. Both the file and the symbolic link are on the same system. By creating this link, you can access the file /etc/termcap using the alternate pathname, /usr/sarah/terms.

```
$ ln —s /etc/termcap /usr/sarah/terms
```

In the following example, the *vi* program on altos1 is referenced by a symbolic link on altos2. After this link is created, when you access /bin/vi on altos2, you are actually accessing /bin/vi on altos1.

```
$ ln —s @altos1/bin/vi @altos2/bin vi
```

NAME

login − sign on

SYNOPSIS

login [name [env-var ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an "end-of-file." (See *How to Get Started* at the beginning of this volume for instructions on how to dial up initially.)

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

```
exec login
```

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure **/etc/profile** is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh*(1)) is initialized, and the file **.profile** in the working directory is executed, if it exists. These specifications are found in the **/etc/passwd** file entry for the user. The name of the command interpreter is − followed by the last component of the interpreter's path name (i.e., **−sh**). If this field in the password file is empty, then the default command interpreter, **/bin/sh** is used. If this field is "*", then a *chroot*(2) is done to the directory named in the directory field of the entry. At that point *login* is re-executed at the new level which must have its own root structure, including **/etc/login** and **/etc/passwd**.

The basic *environment* (see *environ*(5)) is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your login name. The arguments may take either the form *xxx* or *xxx*=*yyy*. Arguments without an equal sign are placed in the environment as

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) in the *UNIX System V Programmer Reference Manual*.

DIAGNOSTICS

Login incorrect if the user name or the password cannot be matched.
No shell, cannot open password file, or *no directory*: consult a UNIX system programming counselor.
No utmp entry. You must exec "login" from the lowest level "sh". if you attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

NAME

logname − get login name

SYNOPSIS

logname

DESCRIPTION

Logname returns the contents of the environment variable **$LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1).
logname(3X), environ(5) in the *UNIX System V Programmer Reference Manual.*

NAME

lorder − find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive *files* (see *ar*(1)). The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1). Note that the link editor (except on the PDP-11) *ld*(1) is capable of multiple passes over an archive in the portable archive format (see *ar*(4)) and does not require that *lorder*(1) be used when building an archive. The usage of the *lorder*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing **.o** files.

```
ar cr library `lorder *.o | tsort`
```

FILES

*symref, *symdef	temporary files

SEE ALSO

ar(1), ld(1), tsort(1).
ar(4) in the *UNIX System V Programmer Reference Manual.*

BUGS

Object files whose names do not end with **.o**, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

NAME

lp, cancel − send/cancel requests to an LP line printer

SYNOPSIS

lp [**−c**] [**−d**dest] [**−m**] [**−n**number] [**−o**option] [**−s**] [**−t**title] [**−w**] files
cancel [ids] [printers]

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name − stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

Lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see *cancel*) or find the status (see *lpstat*(1)) of the request.

The following options to *lp* may appear in any order and may be intermixed with file names:

−c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the **−c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the **−c** option, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

−d*dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see *accept*(1M) and *lpstat*(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).

−m Send mail (see *mail(1))* after the files have been printed. By default, no mail is sent upon normal completion of the print request.

−n*number* Print *number* copies (default of 1) of the output.

−o*option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **−o** keyletter more than once. For more information about what is valid for *options*, see *Models* in *lpadmin*(1M).

−s Suppress messages from *lp*(1) such as "request id is ...".

−t*title* Print *title* on the banner page of the output.

−w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

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

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1).
accept(1M), lpadmin(1M), lpsched(1M) in the *UNIX System V Administrator Reference Manual.*

NAME

lpstat − print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

Lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *Lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

```
−u"user1, user2, user3"
```

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

```
lpstat −o
```

prints the status of all output requests.

−a[*list*]	Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.
−c[*list*]	Print class names and their members. *List* is a list of class names.
−d	Print the system default destination for *lp*.
−o[*list*]	Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.
−p[*list*]	Print the status of printers. *List* is a list of printer names.
−r	Print the status of the LP request scheduler
−s	Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.
−t	Print all status information.
−u[*list*]	Print status of output requests for users. *List* is a list of login names.
−v[*list*]	Print the names of printers and the path names of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

NAME

ls — list contents of directory

SYNOPSIS

ls [**−RadCxmlnogrtucpFbqisf**] [names]

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the **−C** and **−x** options enable multi-column formats, and the **−m** option enables stream output format in 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.

There are an unbelievable number of options:

−R Recursively list subdirectories encountered.

−a List all entries; usually entries whose names begin with a period (.) are not listed.

−d If an argument is a directory, list only its name (not its contents); often used with **−l** to get the status of a directory.

−C Multi-column output with entries sorted down the columns.

−x Multi-column output with entries sorted across rather than down the page.

−m Stream output format.

—L List all files and symbolic links. All symbolic links are listed with a greater-than sign (>) to the right of the filename, for example:

abc def> dir1 dir2>

where **def** and **dir2** are symbolic links.

−l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.

−n The same as **−l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

−o The same as **−l**, except that the group is not printed.

−g The same as **−l**, except that the owner is not printed.

−r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

−t	Sort by time 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).
−c	Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**−t**) or printing (**−l**).
−p	Put a slash (/) after each filename if that file is a 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.
−b	Force printing of non-graphic characters to be in the octal \ddd notation.
−q	Force printing of non-graphic characters in file names as the character (?).
−i	For each file, print the i-number in the first column of the report.
−s	Give size in blocks, including indirect blocks, for each entry.
−f	Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l, −t, −s,** and **−r,** and turns on **−a;** the order is the order in which entries appear in the directory.

The mode printed under the **−l** option consists of 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 (a.k.a. "named pipe") special file;
−	if the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

r	if the file is readable;
w	if the file is writable;
x	if the file is executable;
−	if the indicated permission is *not* granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user-execute permission character is given as **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or **−**) is **t** if the 1000 (octal) bit of the mode is on; see *chmod*(1) for the meaning of this mode. The indications of set-ID and 1000 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**.
/usr/lib/terminfo/*	to get terminal information.

SEE ALSO

chmod(1), find(1).

BUGS

Unprintable characters in file names may confuse the columnar output options.

NAME

m4 − macro processor

SYNOPSIS

m4 [options] [files]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is **−**, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

−e Operate interactively. Interrupts are ignored and the output is unbuffered.

−s Enable line sync output for the C preprocessor (#line ...)

−B*int* Change the size of the push-back and argument collection buffers from the default of 4,096.

−H*int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

−S*int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

−T*int* Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any **−D** or **−U** flags:

−D*name*[=*val*]
Defines *name* to *val* or to null in *val*'s absence.

−U*name*
undefines *name*.

Macro calls have the form:

name(arg1,arg2, ..., argn)

The (must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define	the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of **$***n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the arguments separated by commas; **$@** is like **$***, but each argument is quoted (with the current quotes).
undefine	removes the definition of the macro named in its argument.
defn	returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.
pushdef	like *define*, but saves any previous definition.
popdef	removes current definition of its argument(s), exposing the previous one, if any.
ifdef	if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on UNIX system versions of *m4*.
shift	returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.
changequote	change quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., ` ').
changecom	change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.
divert	*m4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
undivert	causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.
divnum	returns the value of the current output stream.
dnl	reads and discards characters up to and including the next new-line.
ifelse	has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
incr	returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

decr	returns the value of its argument decremented by 1.
eval	evaluates its argument 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. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
len	returns the number of characters in its argument.
index	returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.
substr	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
translit	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	returns the contents of the file named in the argument.
sinclude	is identical to *include*, except that it says nothing if the file is inaccessible.
syscmd	executes the UNIX system command given in the first argument. No value is returned.
sysval	is the return code from the last call to *syscmd*.
maketemp	fills in a string of XXXXX in its argument with the current process ID.
m4exit	causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.
m4wrap	argument 1 will be pushed back at final EOF; example: m4wrap(`cleanup()')
errprint	prints its argument on the diagnostic output file.
dumpdef	prints current names and definitions, for the named items, or for all if no arguments are given.
traceon	with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
traceoff	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 by B. W. Kernighan and D. M. Ritchie.

NAME

pdp11, u3b, u3b5, vax — provide truth value about your processor type

SYNOPSIS

pdp11

u3b

u3b5

vax

m68k

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

pdp11	True if you are on a PDP-11/45 or PDP-11/70.
u3b	True if you are on a 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.
m68k	True if you are on an Altos 68000 or 3068 computer.

The commands that do not apply will return a false (non-zero) value. These commands are often used within *make*(1) makefiles and shell procedures to increase portability.

SEE ALSO

make(1), sh(1), test(1), true(1).

NAME

mail, rmail − send mail to users or read mail

SYNOPSIS

mail [**−epqr**] [**−f** file]

mail [**−t**] persons

rmail [**−t**] persons

DESCRIPTION

Mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a **?**, and a line is read from the standard input to determine the disposition of the message:

<new-line>	Go on to next message.
+	Same as <new-line>.
d	Delete message and go on to next message.
p	Print message again.
−	Go back to previous message.
s [*files*]	Save message in the named *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*.
*	Print a command summary.

The optional arguments alter the printing of the mail:

−e	causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.
−p	causes all mail to be printed without prompting for disposition.
−q	causes *mail* to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.
−r	causes messages to be printed in first-in, first-out order.
−f*file*	causes *mail* to use *file* (e.g., **mbox**) instead of the default *mailfile*.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a **.**) and adds it to each *person*'s *mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., "From ...") are preceded with a **>**. The **−t** option causes the message to be preceded by all *persons* the *mail* is sent to. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file **dead.letter** will be saved to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated every time 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 a recipient's name causes the message to be sent to user **b!cde** on system **a**. System **a** will interpret that destination as a request to send the message to user **cde** on system **b**. This might be useful, for instance, if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**. *Mail* will 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 will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment. In order for forwarding to work properly the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

Rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

FILES

/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).

BUGS

Conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

NAME

mailx — interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

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 (see "MBOX" (ENVIRONMENT VARIABLES) for a description of this file). Messages remain in this file until forcibly removed.

On the command line, *options* start with a dash (—) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the *mailbox*. Command line options are:

—d	Turn on debugging output. Neither particularly interesting nor 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. Overrides the "record" variable, if set (see ENVIRONMENT VARIABLES).
—h *number*	The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops.
—H	Print header summary only.
—i	Ignore interrupts. See also "ignore" (ENVIRONMENT VARIABLES).
—n	Do not initialize from the system default *Mailx.rc* file.
—N	Do not print initial header summary.
—r *address*	Pass *address* to network delivery software. All tilde commands are disabled.
—s *subject*	Set the Subject header field to *subject*.
—u *user*	Read *user*'s *mailbox*. This is only effective if *user*'s *mailbox* is not read protected.
—U	Convert *uucp* style addresses to internet standards. Overrides the "conv" environment variable.

When reading mail, *mailx* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see COMMANDS below). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the **set** and **uns**et commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp*(1) for recording outgoing mail on paper. Alias groups are set by the **a**lias command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, **p**rint is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a 'current' message, marked by a '>' 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**.
.	The current message.
^	The first undeleted message.
$	The last message.
*****	All messages.
n−m	An inclusive range of message numbers.
user	All messages from **user**.
/string	All messages with **string** in the subject line (case ignored).
:*c*	All messages of type *c*, where *c* is one of:

d	deleted messages
n	new messages
o	old messages
r	read messages
u	unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh*(1)). Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* reads commands from a system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. Most regular commands are legal inside start-up files, the most common use being to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, **C**opy, **e**dit, **fo**llowup, **F**ollowup, **ho**ld, **m**ail, **pre**serve, **r**eply, **R**eply, **sh**ell, and **v**isual. Any errors in the start-up file cause the remaining lines in the file to be ignored.

COMMANDS

The following is a complete list of *mailx* commands:

!*shell-command*
: Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).

# *comment*
: Null command (comment). This may be useful in *.mailrc* files.

=
: Print the current message number.

?
: Prints a summary of commands.

alias *alias name ...*
group *alias name ...*
: Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

alternates *name ...*
: Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **alt**ernates prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).

cd [*directory*]
chdir [*directory*]
: Change directory. If *directory* is not specified, $HOME is used.

copy [*filename*]
copy [*msglist*] *filename*
: Copy messages to the file without marking the messages as saved. Otherwise equivalent to the **s**ave command.

Copy [*msglist*]
: Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the Save command.

delete [*msglist*]
: Delete messages from the *mailbox*. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).

discard [*header-field ...*]
ignore [*header-field ...*]
: Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The **P**rint and **T**ype commands override this command.

dp [*msglist*]
dt [*msglist*]
: Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a **d**elete

command followed by a **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" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is *ed*(1).

exit
xit
Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **quit**).

file [*filename*]
folder [*filename*]
Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

%	the current *mailbox*.
%**user**	the *mailbox* for **user**.
#	the previous file.
&	the current *mbox*.

Default file is the current *mailbox*.

folders
Print the names of the files in the directory set by the "folder" variable (see ENVIRONMENT VARIABLES).

followup [*message*]
Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the **F**ollowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

Followup [*msglist*]
Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the **fo**llowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

from [*msglist*]
Prints the header summary for the specified messages.

group *alias name* ...
alias *alias name* ...
Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

headers [*message*]
Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the **z** command.

help

Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]

Holds the specified messages in the *mailbox.*

if *s*|*r*
*mail-command*s
else
*mail-command*s
endif

Conditional execution, where *s* will execute following *mail-command*s, up to an **else** or **endif**, if the program is in *send* mode, and *r* causes the *mail-command*s to be executed only in *receive* mode. Useful in the *.mailrc* file.

ignore *header-field* ...
discard *header-field* ...

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The **P**rint and **T**ype commands override this command.

list

Prints all commands available. No explanation is given.

mail *name* ...

Mail a message to the specified users.

mbox [*msglist*]

Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a description of this file. See also the **ex**it and **q**uit commands.

next [*message*]

Go to next message matching *message.* A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglist*s above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]

Pipe the message through the given *shell-command.* The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [*msglist*]
hold [*msglist*]

Preserve the specified messages in the *mailbox.*

Print [*msglist*]
Type [*msglist*]
: Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command.

print [*msglist*]
type [*msglist*]
: Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

quit
: Exit from *mailx*, storing messages that were read in *mbox* and unread messages in the *mailbox*. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]
: Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

reply [*message*]
respond [*message*]
: Reply to the specified message, including all other recipients of the message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

Save [*msglist*]
: Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the **C**opy, **fo**llowup, and **F**ollowup commands and "outfolder" (ENVIRONMENT VARIABLES).

save [*filename*]
save [*msglist*] *filename*
: Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the **ex**it and **q**uit commands).

set
set *name*
set *name=string*
set *name=number*
: Define a variable called *name*. The variable may be given a null, string, or numeric value. **Se**t by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the *mailx* variables.

shell
: Invoke an interactive shell (see also "SHELL" (ENVIRONMENT VARIABLES)).

size [*msglist*]
: Print the size in characters of the specified messages.

source *filename*
: Read commands from the given file and return to command mode.

top [*msglist*]
: Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

touch [*msglist*]
: Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See **exit** and **quit**.

Type [*msglist*]
Print [*msglist*]
: Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ignore** command.

type [*msglist*]
print [*msglist*]
: Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

undelete [*msglist*]
: Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

unset *name* ...
: Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version
: Prints the current version and release date.

visual [*msglist*]
: Edit the given messages with a screen editor. The messages are placed in a temporary file and the "VISUAL" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

write [*msglist*] *filename*
: Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the **save** command.

xit
exit
: Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **quit**).

z[+|-]
Scroll the header display forward or backward one screen—full. The number of headers displayed is set by the "screen" variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (~). See "escape" (ENVIRONMENT VARIABLES) for changing this special character.

~! *shell-command*
Escape to the shell.

~.
Simulate end of file (terminate message input).

~: *mail-command*
~_ *mail-command*
Perform the command-level request. Valid only when sending a message while reading mail.

~?
Print a summary of tilde escapes.

~A
Insert the autograph string "Sign" into the message (see ENVIRONMENT VARIABLES).

~a
Insert the autograph string "sign" into the message (see ENVIRONMENT VARIABLES).

~b *name* ...
Add the *names* to the blind carbon copy (Bcc) list.

~c *name* ...
Add the *names* to the carbon copy (Cc) list.

~d
Read in the *dead.letter* file. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~e
Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT VARIABLES).

~f [*msglist*]
Forward the specified messages. The messages are inserted into the message, without alteration.

~h
Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~i *string*
Insert the value of the named variable into the text of the message. For example, **~A** is equivalent to '**~i** Sign.'

~m [*msglist*]
: Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~p
: Print the message being entered.

~q
: Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead.letter*. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~r *filename*
~< *filename*
~< !*shell-command*
: Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~s *string* ...
: Set the subject line to *string*.

~t *name* ...
: Add the given *names* to the To list.

~v
: Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

~w *filename*
: Write the partial message onto the given file, without the header.

~x
: Exit as with **~q** except the message is not saved in *dead.letter*.

~| *shell-command*
: Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx*.

HOME=*directory*
: The user's base of operations.

MAILRC=*filename*
: The name of the start-up file. Default is $HOME/.mailrc.

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the **set** command at any time. The **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. Default is **noallnet**. See also the **alt**ernates command and the "metoo" variable.

append
Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend.**

askcc
Prompt for the Cc list after message is entered. Default is **noaskcc**.

asksub
Prompt for subject if it is not specified on the command line with the −s option. Enabled by default.

autoprint
Enable automatic printing of messages after **d**elete and **u**ndelete commands. Default is **noautoprint**.

bang
Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). Default is **nobang**.

cmd=*shell-command*
Set the default command for the **pi**pe command. No default value.

conv=*conversion*
Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also "sendmail" and the **−U** command line option.

crt=*number*
Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable (*pg*(1) by default). Disabled by default.

DEAD=*filename*
The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is $HOME/dead.letter.

debug
Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

dot
Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR=*shell-command*
The command to run when the **e**dit or **~e** command is used. Default is *ed*(1).

escape=*c*
Substitute *c* for the ~ escape character.

folder=*directory*
: The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a *mailx* command line, "folder" must be an exported *sh* environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header
: Enable printing of the header summary when entering *mailx*. Enabled by default.

hold
: Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is **nohold**.

ignore
: Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof
: Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the ~. command. Default is **noignoreeof**. See also "dot" above.

keep
: When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave
: Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*
: The name of the file to save messages which have been read. The **xit** command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

metoo
: If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

LISTER=*shell-command*
: The command (and options) to use when listing the contents of the "folder" directory. The default is *ls*(1).

onehop
: When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder
: Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the path name is absolute. Default is **nooutfolder**. See "folder" above and the Save, Copy, **fo**llowup, and **F**ollowup commands.

page
: Used with the **pi**pe command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER=*shell-command*
: The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg*(1).

prompt=*string*
: Set the *command mode* prompt to *string*. Default is "? ".

quiet
: Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet**.

record=*filename*
: Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save
: Enable saving of messages in *dead.letter* on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen=*number*
: Sets the number of lines in a screen–full of headers for the **h**eaders command.

sendmail=*shell-command*
: Alternate command for delivering messages. Default is *mail*(1).

sendwait
: Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL=*shell-command*
: The name of a preferred command interpreter. Default is *sh*(1).

showto
: When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string*
: The variable inserted into the text of a message when the **~a** (autograph) command is given. No default (see also **~i** (TILDE ESCAPES)).

Sign=*string*
: The variable inserted into the text of a message when the **~A** command is given. No default (see also **~i** (TILDE ESCAPES)).

toplines=*number*
The number of lines of header to print with the **top** command. Default is 5.

VISUAL=*shell-command*
The name of a preferred screen editor. Default is *vi*(1).

FILES

$HOME/.mailrc	personal start-up file
$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO

mail(1), pg(1), ls(1).

BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **un**set.

The full internet addressing is not fully supported by *mailx*. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by *mail*(1) (the standard mail delivery program).

NAME

make − maintain, update, and regenerate groups of programs

SYNOPSIS

make [**−f** makefile] [**−p**] [**−i**] [**−k**] [**−s**] [**−r**] [**−n**] [**−b**] [**−e**] [**−m**] [**−t**] [**−d**] [**−q**] [names]

DESCRIPTION

The following is a brief description of all options and some special names:

−f *makefile*	Description file name. *Makefile* is assumed to be the name of a description file. A file name of − denotes the standard input. The contents of *makefile* override the built-in rules if they are present.
−p	Print out the complete set of macro definitions and target descriptions.
−i	Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.
−k	Abandon work on the current entry, but continue on other branches that do not depend on that entry.
−s	Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.
−r	Do not use the built-in rules.
−n	No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.
−b	Compatibility mode for old makefiles.
−e	Environment variables override assignments within makefiles.
−m	Print a memory map showing text, data, and stack. This option is a no-operation on systems without the *getu* system call.
−t	Touch the target files (causing them to be up-to-date) rather than issue the usual commands.
−d	Debug mode. Print out detailed information on files and times examined.
−q	Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.
.DEFAULT	If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.
.PRECIOUS	Dependents of this target will not be removed when quit or interrupt are hit.
.SILENT	Same effect as the **−s** option.
.IGNORE	Same effect as the **−i** option.

Make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **−f** option is present, **makefile**, **Makefile**, **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is **−**, the standard input is taken. More than one − *makefile* argument pair may appear.

Make updates a target only if its dependents are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a :, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that begin with a tab are shell commands to be executed to update the target. The first line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><new-line> sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o −o pgm
a.o: incl.h a.c
        cc −c a.c
b.o: incl.h b.c
        cc −c b.c
```

Command lines are executed one at a time, each by its own shell. The first one or two characters in a command can be the following: -, @, -@, or @-. If @ is present, printing of the command is suppressed. If - is present, *make* ignores an error. A line is printed when it is executed unless the **−s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the initial character sequence contains a @. The **−n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under *Environment*). The **−t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the **−i** option is present, or the entry **.IGNORE:** appears in *makefile*, or the initial character sequence of the command contains -. the error is ignored. If the **−k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **−b** option allows old makefiles (those written for the old version of *make*) to run without errors. The difference between the old version of *make* and this version is that this version requires all dependency lines to have a (possibly null or implicit) command associated with them. The previous version of *make* assumed, if no command was specified explicitly, that the command was null.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name **.PRECIOUS**.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **−e** option causes the environment to override the macro assignments in a makefile.

The **MAKEFLAGS** environment variable is processed by *make* as containing any legal input option (except **−f**, **−p**, and **−d**) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the **−n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make −n** recursively on a whole software system to see what would have been executed. This is because the **−n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Macros

Entries of the form *string1* = *string2* are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(*string1*[:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

$*** The macro **$* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@ The $@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$< The **$<** macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module which is out-of-date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
        cc −c −O $*.c
```

or:

```
.c.o:
        cc −c −O $<
```

$?** The **$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.

$% The **$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **$@** evaluates to **lib** and **$%** evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros, the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**. The reasons for this are debatable.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c~ .sh .sh~ .c.o .c~.o .c~.c .s.o .s~.o .y.o .y~.o .l.o .l~.o
.y.c .y~.c .l.c .c.a .c~.a .s~.a .h~.h
```

The internal rules for *make* are contained in the source file **rules.c** for the *make* program. These rules can be locally modified. To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
make −fp − 2>/dev/null </dev/null
```

The only peculiarity in this output is the **(null)** string which *printf*(3S) prints when handed a null string.

A tilde in the above rules refers to an SCCS file (see *sccsfile*(4)). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., **.c:**) is the definition of how to build *x* from *x*.**c**. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

```
.SUFFIXES: .o .c .y .l .s
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
pgm: a.o b.o
        cc a.o b.o −o pgm
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library which contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form **.*XX*.a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:     lib(file1.o) lib(file2.o) lib(file3.o)
         @echo lib is now up-to-date
.c.a:
         $(CC) −c $(CFLAGS) $<
         ar rv $@ $*.o
         rm −f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:     lib(file1.o) lib(file2.o) lib(file3.o)
         $(CC) −c $(CFLAGS) $(?:.o=.c)
         ar rv lib $?
         rm $?  @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c˜**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile and s.[Mm]akefile

SEE ALSO

cc(1), cd(1), lex(1), sh(1), yacc(1).
printf(3S), sccsfile(4) in the *UNIX System V Programmer Reference Manual.*

BUGS

Some commands return non-zero status inappropriately; use **−i** to overcome the difficulty. File names with the characters = : @ will 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 illegal. You cannot build **lib(file.o)** from **file.o**. The macro **$(a:.o=.c˜)** does not work.

NAME

makekey — generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

Makekey is intended for programs that perform encryption (e.g., *ed*(1) and *crypt*(1)). Usually, its input and output will be pipes.

SEE ALSO

crypt(1), ed(1).
passwd(4) in the *UNIX System Programmer Reference Manual.*

NAME

man − print entries in this manual

SYNOPSIS

man [options] [section] titles

DESCRIPTION

Man locates and prints the entry of this manual named *title* in the specified *section*. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The *title* is entered in lower case. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for *title* and all occurrences of it are printed. *Options* and their meanings are:

−T*term*	Print the entry as appropriate for terminal type *term*. For a list of recognized values of *term*, type **help term2**. The default value of *term* is **450**.
−w	Print on the standard output only the *path names* of the entries, relative to **/usr/man**, or to the current directory for **−d** option.
−d	Search the current directory rather than **/usr/catman**; requires the full file name (e.g., **cu.1c**, rather than just **cu**).
−c	Causes *man* to invoke *col*(1); note that *col*(1) is invoked automatically by *man* unless *term* is one of **300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620**, and **X**.

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 **−Tlp** when sending the output of *man* to a line printer.

Section may be changed before each *title*.

As an example:

```
man man
```

would reproduce on the terminal this entry, as well as any other entries named *man* that may exist in other sections of the manual.

FILES

/usr/catman/?_man/man[1-8]/*	Preformatted manual entries

SEE ALSO

term(5) in the *UNIX System V Programmer Reference Manual.*

CAVEAT

The *man* command prints manual entries that were formatted by *nroff* when the UNIX system was installed. Entries are originally formatted with terminal type 37, and are printed using the correct terminal filters as derived from the −T*term* and **$TERM** settings. Typesetting or other non-standard printing of manual entries requires installation of the UNIX system Documenter's Workbench.

NAME

mesg − permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir − make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (possibly altered by *umask*(1)). Standard entries, ., for the directory itself, and .., for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

sh(1), rm(1), umask(1).

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

NAME

netlet—execute a command on remote system without access permissions for that system

SYNOPSIS

netlet machine1 machine2 ... **—c** command arguments

DESCRIPTION

The *netlet* command executes commands on remote systems even if you do not have permission to access the remote systems.

Before you use *netlet*, make sure that:

1. The command you use with *netlet* exists as an entry in the /etc/net/cmd directory on each system.
2. Each command in the /etc/net/cmd directory on each system is linked to the same file. This ensures that all systems agree on the version of the command to be used.

EXAMPLES

The following commands link the *tar* command on each system to the same file, /bin/tar, on System B to make sure the same version of the command is being run.

To create the appropriate entries in the /etc/net/cmd directory, log in as root on any system, and enter:

```
mkdir @A/etc/net/cmd
mkdir @B/etc/net/cmd
mkdir @C/etc/net/cmd
ln —s @B/bin/tar @A/etc/net/cmd/tar
ln —s @B/bin/tar @B/etc/net/cmd/tar
ln —s @B/bin/tar @C/etc/net/cmd/tar
```

Note that you must have access permissions on systems A, B, and C to create these files, but you need not have access permissions to use *netlet*.

Then, you can use *netlet* to run *tar* on a remote system:

```
$ netlet @C —c tar cv <CR>
```

NAME

newform − change the format of a text file

SYNOPSIS

newform [−s] [−itabspec] [−otabspec] [−bn] [−en] [−pn] [−an] [−f] [−cchar] [−ln] [files]

DESCRIPTION

Newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for **−s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "−e15 −l60" will yield results different from "−l60 −e15". Options are applied to all *files* on the command line.

−i*tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* may be −−, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec*(4)). If no *tabspec* is given, *tabspec* defaults to **−8**. A *tabspec* of **−0** expects no tabs; if any are found, they are treated as **−1**.

−o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **−i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **−8**. A *tabspec* of **−0** means that no spaces will be converted to tabs on output.

−l*n* Set the effective line length to *n* characters. If *n* is not entered, **−l** defaults to 72. The default line length without the **−l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **−i** to expand tabs to spaces).

−b*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **−l***n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **−b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform −l1 −b7 file-name
```

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.

−e*n* Same as **−b***n* except that characters are truncated from the end of the line.

−c*k* Change the prefix/append character to *k*. Default character for *k* is a space.

−p*n* Prefix *n* characters (see **−c***k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

−a*n* Same as **−p***n* except characters are appended to the end of a line.

−f Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last*

−o option. If no **−o** option is specified, the line which is printed will contain the default specification of **−8**.

−s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

```
newform −s −i −l −a −e file-name
```

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*Newform* was called with a bad option.
not −s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

EXIT CODES

0 − normal execution
1 − for any error

SEE ALSO

csplit(1), tabs(1).
fspec(4) in the *UNIX System V Programmer Reference Manual.*

BUGS

Newform normally only keeps track of physical characters; however, for the **−i** and **−o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

Newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **−i− −** or **−o− −**).

If the **−f** option is used, and the last **−o** option specified was **−o− −**, and was preceded by either a **−o− −** or a **−i− −**, the tab specification format line will be incorrect.

NAME

netstat — display status of systems in WorkNet

SYNOPSIS

netstat [machine1 machine2 ...]

DESCRIPTION

The *netstat* command tells you the status of all or some of the systems in WorkNet.

EXAMPLES

To find out what computers on the network you have access to and their status, enter:

$ netstat <CR>

The status of all systems on your network then appears:

```
SYSTEM      STATUS          LOAD        ACCESS
machine1    local system                permitted
machine2    not available               not permitted
machine3    available       10          not permitted
```

To display the status for a specific system, enter:

$ netstat machine2 <CR>

where: machine2 is the system name

The status of the system machine2 then appears:

```
SYSTEM      STATUS          LOAD        ACCESS
machine2    not available               not permitted
```

SEE ALSO

netlet(1)

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 **$** (default) and has not exported **PS1**. After an invocation of *newgrp* , successful or not, their **PS1** will now be set to the default prompt string **$**. Note that the shell command *export* (see *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry.

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) in the *UNIX System V Programmer Reference Manual.*

BUGS

There is no convenient way to enter a password into **/etc/group**. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

news − print news items

SYNOPSIS

news [**−a**] [**−n**] [**−s**] [items]

DESCRIPTION

News is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *News* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

The **−a** option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

The **−n** option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

The **−s** option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(4), environ(5) in the *UNIX System V Programmer Reference Manual.*

NAME

nice − run a command at low priority

SYNOPSIS

nice [−increment] command [arguments]

DESCRIPTION

Nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., −**−10**.

SEE ALSO

nohup(1).
nice(2) in the *UNIX System V Programmer Reference Manual.*

DIAGNOSTICS

Nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

NAME

nl — line numbering filter

SYNOPSIS

nl [**−h**type] [**−b**type] [**−f**type] [**−v**start#] [**−i**incr] [**−p**] [**−l**num] [**−s**sep] [**−w**width] [**−n**format] [**−d**delim] file

DESCRIPTION

Nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

Nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

−b*type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **p***string*, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is **t** (text lines numbered).

−h*type* Same as **−b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

−f*type* Same as **−b***type* except for footer. Default for logical page footer is **n** (no lines numbered).

−p Do not restart numbering at logical page delimiters.

−v*start#* *Start#* is the initial value used to number logical page lines. Default is **1**.

−i*incr* *Incr* is the increment value used to number logical page lines. Default is **1**.

−s*sep* *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

−w*width* *Width* is the number of characters to be used for the line number. Default *width* is **6**.

−n*format* *Format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes supressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

−l*num* *Num* is the number of blank lines to be considered as one. For example, **−l2** results in only the second adjacent blank being numbered (if the appropriate **−ha**, **−ba**, and/or **−fa** option is set). Default is **1**.

−d*xx* The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **−d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl −v10 −i10 −d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

NAME

nm − print name list of common object file

SYNOPSIS

nm **[−o]** **[−x]** **[−h]** **[−v]** **[−n]** **[−e]** **[−f]** **[−u]** **[−V]** **[−T]** file-names

DESCRIPTION

The *nm* command displays the symbol table of each common object file *file-name*. *File-name* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

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 or of a union then the structure or union tag will be given following the type (e.g., struct-tag). If the symbol is an array, then the array dimensions will be given following the type (e.g., **char[n][m]**). Note that the object file must have been compiled with the **−g** option of the *cc*(1) command for this information to appear.
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 appear.
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 appear.
Section	For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).

The output of *nm* may be controlled using the following options:

−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	Do not display the output header data.
−v	Sort external symbols by value before they are printed.
−n	Sort external symbols by name before they are printed.
−e	Print only external and static symbols.
−f	Produce full output. Print redundant symbols (.text, .data and .bss), normally suppressed.
−u	Print undefined symbols only.
−V	Print the version of the nm command executing on the standard error output.
−T	By default, *nm* prints the entire name of the symbols listed. Since 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 will overflow 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.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **nm name −e −v** and **nm −ve name** print the static and external symbols in *name*, with external symbols sorted by value.

FILES

/usr/tmp/nm??????

CAVEATS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **−v** and **−n** options should be used only in conjunction with the **−e** option.

SEE ALSO

as(1), cc(1), ld(1).
a.out(4), ar(4) in the *UNIX System V Programmer Reference Manual.*

DIAGNOSTICS

"nm: name: cannot open"
if *name* cannot be read.

"nm: name: bad magic"
if *name* is not an appropriate common object file.

"nm: name: no symbols"
if the symbols have been stripped from *name*.

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup command [arguments]

DESCRIPTION

Nohup executes *command* with hangups and quits ignored. If output is not re-directed by the user, both standard output and standard error are sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see *sh*(1)):

```
nohup file &
```

An example of what the contents of *file* could be is:

```
tbl ofile | eqn | nroff > nfile
```

SEE ALSO

chmod(1), nice(1), sh(1).
signal(2) in the *UNIX System V Programmer Reference Manual.*

WARNINGS

nohup command1; command2   *nohup* applies only to *command1*
nohup (command1; command2) is syntactically incorrect.

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/1m&
```

while

```
nohup cpio –o <list >/dev/rmt/1m 2>errors&
```

puts the error messages into file *errors*.

NAME

od − octal dump

SYNOPSIS

od [**−bcdosx**] [file] [[**+**]offset[**.**][**b**]]

DESCRIPTION

Od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **−o** is default. The meanings of the format options are:

−b	Interpret bytes in octal.
−c	Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=**\0**, backspace=**\b**, form-feed=**\f**, new-line=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers.
−d	Interpret words in unsigned decimal.
−o	Interpret words in octal.
−s	Interpret 16-bit words in signed decimal.
−x	Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If **.** is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by **+**.

Dumping continues until end-of-file.

SEE ALSO

dump(1).

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [**–f**] name ...

pcat name ...

unpack name ...

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name*.**z** with the same access modes, access and modified dates, and owner as those of *name*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

Pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

> the file appears to be already packed;
> the file name has more than 12 characters;
> the file has links;
> the file is a directory;
> the file cannot be opened;
> no disk storage blocks will be saved by packing;
> a file called *name*.**z** already exists;
> the **.z** file cannot be created;
> an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(1) does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.**z** use:

```
        pcat name.z
or just:
        pcat name
```

To make an unpacked copy, say *nnn*, of a packed file named *name*.**z** (without destroying *name*.**z**) use the command:

```
pcat name >nnn
```

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).

NAME

passwd – change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This 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 ID is zero is called a super-user; see *id*(1), and *su*(1). Super-users may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd

SEE ALSO

login(1), id(1), su(1).
crypt(3C), passwd(4) in the *UNIX System V Programmer Reference Manual.*

NAME

paste − merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste **−d**list file1 file2 ...
paste **−s** [**−d**list] file1 file2 ...

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if − is used in place of a file name.

The meanings of the options are:

−d Without this option, the new-line characters of each but the last file (or last line in case of the **−s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list One or more characters immediately following **−d** replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no **−s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), \\ (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use *−d*"\\\\").

−s Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **−d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.

− May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

ls | paste −d" " −	list directory in one column
ls | paste − − − −	list directory in four columns
paste −s −d"\t\n" file	combine pairs of lines into lines

SEE ALSO

cut(1), grep(1), pr(1).

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except for **−s** option, no more than 12 input files may be specified.

NAME

pg − file perusal filter for soft-copy 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 soft-copy terminal. (The file name − and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo*(4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

−number
: An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

−p *string*
: Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

−c Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo*(4) data base.

−e Causes *pg not* to pause at the end of each file.

−f Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The *−f* option inhibits *pg* from splitting lines.

−n Normally, commands must be terminated by a *<newline>* character. This option causes an automatic end of command as soon as a command letter is entered.

−s Causes *pg* to print all messages and prompts in standout mode (usually inverse video).

+linenumber
: Start up at *linenumber*.

+/pattern/
: Start up at the first line containing the regular expression pattern.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<*newline*> or <*blank*>
: This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
: Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L** Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(1) are available. They must always be terminated by a <*newline*>, even if the *−n* option is specified.

i/pattern/
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*
i?pattern?
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w** Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
: Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a <*newline*>, even if the *−n* option is specified.

h Help by displaying an abbreviated summary of available commands.

q or **Q** Quit *pg*.

!*command*

Command is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a <*newline*>, even if the −*n* option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK**, **DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

1027.sp40u

/usr/lib/terminfo/*
Terminal information data base

/tmp/pg*
Temporary file when input is from a pipe

SEE ALSO

crypt(1), ed(1), grep(1).
terminfo(4) in the *UNIX System V Programmer Reference Manual.*

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options (e.g., *crypt*(1)), terminal settings may not be restored correctly.

NAME

pr − print files

SYNOPSIS

pr [options] [files]

DESCRIPTION

Pr prints the named files on the standard output. If *file* is −, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **−s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

The below *options* may appear singly or be combined in any order:

+*k* Begin printing with page *k* (default is 1).

−*k* Produce *k*-column output (default is 1). The options **−e** and **−i** are assumed for multi-column output.

−a Print multi-column output across the page.

−m Merge and print all files simultaneously, one per column (overrides the −*k*, and **−a** options).

−d Double-space the output.

−e*ck* Expand *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).

−i*ck* In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

−n*ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **−m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

−w*k* Set the width of a line to *k* character positions (default is 72 for equal-width multi-column output, no limit otherwise).

−o*k* Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

−l*k* Set the length of a page to *k* lines (default is 66).

−h Use the next argument as the header to be printed instead of the file name.

−p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

−f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

−r Print no diagnostic reports on failure to open files.

−t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

−s*c* Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr −3dh "file list" file1 file2
```

Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr −e9 −t <file1 >file2
```

FILES

/dev/tty*	to suspend messages

SEE ALSO

cat(1).

NAME

prof — display profile data

SYNOPSIS

prof [−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.

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.

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's 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)).

FILES

mon.out	for profile
a.out	for namelist

SEE ALSO

cc(1).
exit(2), profil(2), monitor(3C), prof(5) in the *UNIX System V Programmer Reference Manual.*

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 blur the data. In rare cases, the clock ticks initiating recording of the program counter may "beat" with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely, however.

BUGS

Only programs that call *exit*(2) or return from *main* will cause a profile file to be produced, unless a final call to monitor is explicitly coded.

The use of the **−p** option *cc*(1) to invoke profiling imposes a limit of 600 (300 on the PDP-11) 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 will be overwritten and the **mon.out** file will be corrupted. The number of call counters used will be reported automatically by the *prof* command whenever the number exceeds 5/6 of the maximum.

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 *keyletter* arguments, and file names.

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

−d[*dataspec*]	Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.
−r[*SID*]	Used to specify the *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** keyletter 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** keyletter or the date given by the **−c** option.
−c[*date-time*]	The cutoff date-time **−c**[cutoff]] is 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 2-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** keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(4)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords.
A tab is specified by **\t** and carriage return/new-line is specified by **\n**. 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 keyletter argument allowed to be used with the *special case* is the **−a** keyletter.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1).
sccsfile(4) in the *UNIX System V Programmer Reference Manual.*

Source Code Control System User Guide in the *UNIX System V User Guide.*

DIAGNOSTICS

Use *help*(1) for explanations.

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 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:

−e	Print information about all processes.
−d	Print information about all processes, except process group leaders.
−a	Print information about all processes, except process group leaders and processes not associated with a terminal.
−f	Generate a *full* listing. (See below for meaning of columns in a full listing).
−l	Generate a *long* listing. See below.
−c *corefile*	Use the file *corefile* in place of **/dev/mem**.
−s *swapdev*	Use the file *swapdev* in place of **/dev/swap**. This is useful when examining a *corefile*; a *swapdev* of **/dev/null** will cause the user block to be zeroed out.
−n *namelist*	The argument will be taken as the name of an alternate system *namelist* file in place of **/unix**.
−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's file name (e.g., **tty04**) or if the device's file name starts with **tty**, just the digit identifier (e.g., **04**).
−p *proclist*	Restrict listing to data about processes whose process ID numbers are given in *proclist*.
−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 will be printed unless the **−f** option is used, in which case the login name will be printed.
−g *grplist*	Restrict listing to data about processes whose process group leaders are given in *grplist*.

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 will be listed.

F	(l)	Flags (octal and additive) associated with the process:
		0 swapped;
		1 in core;
		2 system process;
		4 locked-in core (e.g., for physical I/O);
		10 being swapped;
		20 being traced by another process;
		40 another tracing flag;
		100 3B 20 computer: swapin segment expansion; VAX-11/780: text pointer valid;
		200 3B 20 computer: process is child (during fork swap); VAX-11/780: process is partially swapped.

S	(l)	The state of the process:
		0 non-existent;
		S sleeping;
		W waiting;
		R running;
		I intermediate;
		Z terminated;
		T stopped;
		X growing.

UID	(f,l)	The user ID number of the process owner; the login name is printed under the **−f** option.
PID	(all)	The process ID of the process; it is possible to kill a process if you know this datum.
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling.
PRI	(l)	The priority of the process; higher numbers mean lower priority.
NI	(l)	Nice value; used in priority computation.
ADDR	(l)	The memory address of the process (a pointer to the segment table array on the 3B 20 computer), if resident; otherwise, the disk address.
SZ	(l)	The size in blocks of the core image of the process.
WCHAN	(l)	The event for which the process is waiting or sleeping; if blank, the process is running.
STIME	(f)	Starting time of the process.
TTY	(all)	The controlling terminal for the process.
TIME	(all)	The cumulative execution time for the process.
CMD	(all)	The command name; the full command name and its arguments are printed under the **−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).

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

NAME

ptx − permuted index

SYNOPSIS

ptx [options] [input [output]]

DESCRIPTION

Ptx generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of each line. *Ptx* output is in the form:

```
.xx "tail" "before keyword" "keyword and after" "head"
```

where **.xx** is assumed to be an *nroff* or *troff*(1) macro provided by the user, or provided by the *mptx*(5) macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following *options* can be applied:

−f	Fold upper and lower case letters for sorting.
−t	Prepare the output for the phototypesetter.
−w *n*	Use the next argument, *n*, as the length of the output line. The default line length is 72 characters for *nroff* and 100 for *troff*.
−g *n*	Use the next argument, *n*, as the number of characters that *ptx* will reserve in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.
−o *only*	Use as keywords only the words given in the *only* file.
−i *ignore*	Do not use as keywords any words given in the *ignore* file. If the **−i** and **−o** options are missing, use **/usr/lib/eign** as the *ignore* file.
−b *break*	Use the characters in the *break* file to separate words. Tab, newline, and space characters are *always* used as break characters.
−r	Take any leading non-blank characters of each input line to be a reference identifier (as to a page or chapter), separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx*.

FILES

/bin/sort
/usr/lib/eign
/usr/lib/tmac/tmac.ptx

SEE ALSO

nroff(1), troff(1).
mm(5), mptx(5) in the *UNIX System V Programmer Reference Manual.*

BUGS

Line length counts do not account for overstriking or proportional spacing.
Lines that contain tildes (~) are botched, because *ptx* uses that character internally.

NAME

pwd − working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to a UNIX system programming counselor.

NAME

ratfor – rational Fortran dialect

SYNOPSIS

ratfor [options] [files]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:
```
{ statement; statement; statement }
```

decision-making:
```
if (condition) statement [ else statement ]
switch (integer value) {
        case integer:     statement
        ...
        [ default: ]      statement
}
```

loops:
```
while (condition) statement
for (expression; condition; expression) statement
do limits statement
repeat statement [ until (condition) ]
break
next
```

and some syntactic sugar to make programs easier to read and write:

free form input:
multiple statements/line; automatic continuation

comments:
```
# this is a comment.
```

translation of relationals:
>, >=, etc., become **.GT.**, **.GE.**, etc.

return expression to caller from function:
return (expression)

define:
define *name replacement*

include:
include *file*

The option **–h** causes quoted strings to be turned into **27H** constructs. The **–C** option copies comments to the output and attempts to format it neatly. Normally, continuation lines are marked with a **&** in column 1; the option **–6x** makes the continuation character **x** and places it in column 6.

Ratfor is best used with *f77*(1).

SEE ALSO

efl(1), f77(1).

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

NAME

recover—restores the contents of a file system from streaming tape to disk

SYNOPSIS

recover [–i] [–s] mag-tape file-system

DESCRIPTION

The *recover* command copies the tape (specified by mag-tape) to the hard disk file system (specified by file-system). It restores the contents of the tape created by the archive command.

OPTIONS

-i Displays the character string that was specified by the –i option on the archive command that created the tape. This option does not copy the contents of the tape to the hard disk.

-s Displays information from the header block: the number of the tape, the creation date, the starting block number on the tape, and the name of the file system. This option does not copy the contents of the tape to the hard disk.

When restoring the root file system on the hard disk (i.e., /dev/rhd0b), boot UNIX from the Root File System floppy disk.

Be sure to specify /dev/rsct (for streaming tape) when running *archive* or *recover.*

EXAMPLES

/etc/umount /dev/hdla
archive /dev/rhdla /dev/rsct

This command backs up the second hard disk to tape.

/etc/umount /dev/hdla
recover /dev/rsct /dev/rhdla

This command restores the files on the archive tape to the second hard disk.

SEE ALSO

archive(1)

NAME

regcmp − regular expression compile

SYNOPSIS

regcmp [−] files

DESCRIPTION

Regcmp, in most cases, precludes the need for calling *regcmp*(3X) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file*.**i**. If the − option is used, the output will be placed in *file*.**c**. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *File*.**i** files may thus be *included* into C programs, or *file*.**c** files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLES

```
name    "([A−Za−z][A−Za−z0−9_]*)$0"

telno   "\({0,1}([2−9][01][1−9])$0\){0,1} *"
        "([2−9][0−9]{2})$1[ −]{0,1}"
        "([0−9]{4})$2"
```

In the C program that uses the *regcmp* output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

SEE ALSO

regcmp(3X) in the *UNIX System V Programmer Reference Manual*.

NAME

rm, rmdir — remove files or directories

SYNOPSIS

rm [**−fri**] file ...

rmdir dir ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with **y** the file is deleted, otherwise the file remains. No questions are asked when the **−f** option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument **−r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **−i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **−r**, whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

unlink(2) in the *UNIX System V Programmer Reference Manual.*

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file **..** merely to avoid the antisocial consequences of inadvertently doing something like:

```
rm −r .*
```

NAME

rmdel − remove a delta from an SCCS file

SYNOPSIS

rmdel −**r**SID files

DESCRIPTION

Rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see *get*(1)) exists for the named SCCS file, the specified must *not* appear in any entry of the *p-file*).

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

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User Guide*. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES

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

SEE ALSO

delta(1), get(1), help(1), prs(1).
sccsfile(4) in the *UNIX System V Programmer Reference Manual.*

Source Code Control System User Guide in the *UNIX System V User Guide*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

run—run a program on a remote WorkNet system

SYNOPSIS

run machine-name command

DESCRIPTION

The WorkNet *run* command executes a program directly on a remote processor while all terminal input and output is still done using your local terminal.

It may be useful to execute a program on another system if:

- o The other computer has a special version of the operating system which is not in use on yours.
- o The other computer has more memory than yours, and the program you want to run requires this extra memory.

EXAMPLES

In the following example, the *run* command is used to run the process status (*ps*) program on the machine named altos2. The *ps* program then outputs to your local terminal screen the status of all processes running on altos2:

```
$ run altos2 ps —ef <CR>
```

In the next example, the *run* command is used to connect your terminal to the C-shell command line interpreter on another computer:

```
$ run altos2 csh <CR>
```

FILES

/etc/net/runserver

NAME

sact − print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the **−e** option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of -- is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1	specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	specifies the SID for the new delta to be created.
Field 3	contains the logname of the user who will make the delta (i.e., executed a *get* for editing).
Field 4	contains the date that **get −e** was executed.
Field 5	contains the time that **get −e** was executed.

SEE ALSO

delta(1), get(1), unget(1).

DIAGNOSTICS

Use *help*(1) for explanations.

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 is available). These *options* are passed through to *sar*:

−s *time*	Select data later than *time* in the form hh[:mm]. Default is 08:00.
−e *time*	Select data up to *time*. Default is 18:00.
−i *sec*	Select data at intervals as close as possible to *sec* seconds.
−f *file*	Use *file* as the data source for *sar*. Default is the current daily data file **/usr/adm/sa/sa***dd*.

Other *options*:

−T *term*	Produce output suitable for terminal *term*. See *tplot*(1G) for known terminals. If *term* is **vpr**, output is processed by **vpr −p** and queued to a Versatec printer. Default for *term* is **$TERM**.
−x *spec*	x axis specification with *spec* in the form: "name[op name]...[lo hi]"
−y *spec*	y axis specification with *spec* in the same form as above.

Name is either a string that will match a column header in the *sar* report, with an optional device name in square brackets, e.g., **r+w/s[dsk−1]**, or an integer value. *Op* is **+ − *** or **/** surrounded by blanks. Up to five names may be specified. Parentheses are not recognized. Contrary to custom, **+** and **−** have precedence over ***** and **/**. Evaluation is left to right. Thus A / A + B * 100 is evaluated (A/(A+B))*100, and A + B / C + D is (A+B)/(C+D). *Lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, *time* is used. Up to 5 *spec*'s separated by ; may be given for **−y**. Enclose the **−x** and **−y** arguments in "" if blanks or \<CR> are included. The **−y** default is:

−y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

EXAMPLES

To see today's CPU utilization:

```
sag
```

To see activity over 15 minutes of all disk drives:

```
TS=`date +%H:%M`
sar −o tempfile 60 15
TE=`date +%H:%M`
sag −f tempfile −s $TS −e $TE −y "r+w/s[dsk]"
```

FILES

/usr/adm/sa/sa*dd*   daily data file for day *dd*.

SEE ALSO

sar(1), tplot(1G).

NAME

sar − system activity reporter

SYNOPSIS

sar [**−ubdycwaqvmA**] [**−o** file] t [n]

sar [**−ubdycwaqvmA**] [**−s** time] [**−e** time] [**−i** sec] [**−f** file]

DESCRIPTION

Sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the **−o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, **sar** extracts data from a previously recorded *file,* either the one specified by **−f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd.* The starting and ending times of the report can be bounded via the **−s** and **−e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The **−i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

−u Report CPU utilization (the default):
%usr, %sys, %wio, %idle − portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.

−b Report buffer activity:
bread/s, bwrit/s − transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s − accesses of system buffers;
%rcache, %wcache − cache hit ratios, e. g., 1 − bread/lread;
pread/s, pwrit/s − transfers via raw (physical) device mechanism.

−d Report activity for each block device, e. g., disk or tape drive. When data is displayed, the device specification *dsk-* is generally used to represent a disk drive. (On Digital Equipment Corporation machines, the device specification *dsk-* is used to represent a MASSBUS disk, while the specification *dskR-* is used to represent an RA disk.) The device specification used to represent a tape drive is machine dependent. The activity data reported is:
%busy, avque − portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s − number of data transfers from or to device, number of bytes transferred in 512-byte units;
avwait, avserv − average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

−y Report TTY device activity:
rawch/s, canch/s, outch/s − input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s − receive, transmit and modem interrupt rates.

−c Report system calls:
scall/s − system calls of all types;
sread/s, swrit/s, fork/s, exec/s − specific system calls;
rchar/s, wchar/s − characters transferred by read and write system calls.

−w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s − number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs);
pswch/s − process switches.

−a Report use of file access system routines:
iget/s, namei/s, dirblk/s.

−q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc − run queue of processes in memory and runnable;
swpq-sz, %swpocc − swap queue of processes swapped out but ready to run.

−v Report status of 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 today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
sar −o temp 60 10
```

To later review disk and tape activity from that period:

```
sar −d −f temp
```

FILES

/usr/adm/sa/sa*dd* daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G).
sar(1M) in the *UNIX System V Administrator Reference Manual.*

NAME

sccsdiff − compare two versions of an SCCS file

SYNOPSIS

sccsdiff **−r**SID1 **−r**SID2 [**−p**] [**−sn**] files

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

−r*SID?*	*SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.
−p	pipe output for each file through *pr*(1).
−s*n*	*n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1).

Source Code Control System User Guide in the *UNIX System V User Guide*.

DIAGNOSTICS

"*file*: No differences"	If the two versions are the same.

Use *help*(1) for explanations.

NAME

sdb − symbolic debugger

SYNOPSIS

sdb [−w] [−**W**] [objfil [corfil [directory-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* will be limited, but the file can still be examined and the program debugged. The default for *objfil* is **a.out**. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is **core**. The core file need not be present. A − in place of *corfil* will force *sdb* to ignore any core image file. 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** flag.

Names of variables are written just as they are in C or F77. Note that names in C are now of arbitrary length, *sdb* will no longer 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* will interpret a structure as a set of variables. Thus, *sdb* will display the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value *sdb* displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as *variable*[*number*][*number*]..., or as *variable*[*number*,*number*,...]. In place of *number*, the form *number*;*number* may be used to indicate a range of values, * may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, *sdb* displays all the values of an array or 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 *file-name:number* or *procedure:number*. In either case the number is relative to the beginning of the file. If no procedure or file name is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under *sdb*, all addresses refer to the executing program; otherwise they refer to *objfil* or *corfil*. An initial argument of **−w** permits overwriting locations in *objfil*.

Addresses

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

> *b1*address<e1
>
> *file address=address+f1 −b1*

otherwise

> *b2*address<e2
>
> *file address=address+f2 −b2,*

otherwise, the requested *address* is not legal. In some cases (e.g., for programs with separated I and D space) the two segments for a file may overlap.

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

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

Commands

The commands for examining data in the program are:

t Print a stack trace of the terminated or 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)

Legal 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's 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 instruction 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's 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. Since this command is done by software, it can be very slow.

level **v**
: Toggle verbose mode, for use when single stepping with **S**, **s** or **m**. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each C source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A **v** turns verbose mode off if it is on for any level.

k Kill the 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) in the *UNIX System V Programmer Reference Manual.*

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.

BUGS

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

The default type for printing F77 parameters is incorrect. Their address is printed instead of their value.

Tracebacks containing F77 subprograms with multiple entry points may print too many arguments in the wrong order, but their values are correct.

The range of an F77 array subscript is assumed to be *1* to *n*, where *n* is the dimension corresponding to that subscript. This is only significant when the user omits a subscript, or uses * to indicate the full range. There is no problem in general with arrays having subscripts whose lower bounds are not 1.

On the 3B 20 computer there is no hardware trace mode and single-stepping is implemented by setting pseudo breakpoints where possible. This is slow. The *s*, *S*, *i*, and *I* commands do not always convert on the 3B 20 computer due to pseudo-breakpointing. Thus *sdb* will not allow single-stepping from an *indirect* jump, a *switch* instruction, or a *switdt* instruction.

The entry point to an optimized function cannot be found on the 3B 20 computer. Setting a breakpoint at the beginning of an optimized function may cause the middle of some instruction within the function to be overwritten. This problem can be circumvented by disassembling the first few instructions of the function, and manually setting a breakpoint at the first instruction after the stack pointer is adjusted.

NAME

sdiff − side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

Sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x    |    y
a         a
b    <
c    <
d         d
     >    c
```

The following options exist:

−w *n*	Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.
−l	Only print the left side of any lines that are identical.
−s	Do not print identical lines.
−o *output*	Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

l	append the left column to the output file
r	append the right column to the output file
s	turn on silent mode; do not print identical lines
v	turn off silent mode
e l	call the editor with the left column
e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

sed — stream editor

SYNOPSIS

sed [**−n**] [**−e** script] [**−f** sfile] [files]

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **−f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **−e** option and no **−f** options, the flag **−e** may be omitted. The **−n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **−n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of *ed*(1) modified thus:

In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

The escape sequence **\n** matches a new-line *embedded* in the pattern space.

A period **.** matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a**\
text Append. Place *text* on the output before reading the next input line.

(2) **b** *label* Branch to the : command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) **c**\
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) **d** Delete the pattern space. Start the next cycle.

(2) **D** Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

(2) **g** Replace the contents of the pattern space by the contents of the hold space.

(2) **G** Append the contents of the hold space to the pattern space.

(2) **h** Replace the contents of the hold space by the contents of the pattern space.

(2) **H** Append the contents of the pattern space to the hold space.

(1) **i**\
text Insert. Place *text* on the standard output.

(2) **l** List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.

(2) **n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) **N** Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) **p** Print. Copy the pattern space to the standard output.

(2) **P** Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) **q** Quit. Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s**/*regular expression*/*replacement*/*flags*

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed*(1). *Flags* is zero or more of:

n n= 1 - 512. Substitute for just the n th occurrence of the *regular expression.*

g Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Print the pattern space if a replacement was made.

w *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label* Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.

(2) **w** *wfile* Write. Append the pattern space to *wfile*.

(2) **x** Exchange the contents of the pattern and hold spaces.

(2) **y**/*string1*/*string2*/

Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! *function*
: Don't. Apply the *function* (or group, if *function* is {) only to lines *not* selected by the address(es).

(0): *label* This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1) = Place the current line number on the standard output as a line.

(2) { Execute the following commands through a matching } only when the pattern space is selected.

(0) An empty command is ignored.

(0) # If a # appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the # is an 'n', then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

SEE ALSO

awk(1), ed(1), grep(1).

NAME

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. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (| |) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a *simple-command* or one of the following. Unless otherwise stated, the value returned by a command is that of the last *smple-command* executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**
: Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**
: A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation*) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a sub-shell.

{*list*;}

list is simply executed.

name () {*list*;}

Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # 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 writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter*}

The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or @, all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter*:−*word*}

If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}

If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*: +*word*}
: If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
-	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh*.

CDPATH
The search path for the *cd* command.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default "**$** ".

PS2 Secondary prompt string, by default "**>** ".

IFS Internal field separators, normally **space**, **tab**, and **new-line**.

SHACCT
If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. 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 \\, \`, ", and **$**. **"$*"** is equivalent to **"$1 $2** ...", whereas **"$@"** is equivalent to **"$1" "$2"** ....

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple-command* or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<word	Use file *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	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\new-line** is ignored, and \ must be used to quote the characters \, **$**, **`**, and the first character of *word*. If − is appended to **<<**, all leading tabs are stripped from *word* and from the document.
<&digit	Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using **>&digit**.
<&−	The standard input is closed. Similarly for the standard output using **>&−**.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

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's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd                                  and
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **−k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set −k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1, $2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign 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 will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (**:**). The default path is **<null>** (specifying the

current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by *rsh*.

echo [*arg* ...]
: Echo arguments. See *echo*(1) for usage and description.

eval [*arg* ...]
: The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
: The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
: Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
: The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, 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*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
: Equivalent to **exec newgrp** *arg* .... See *newgrp*(1) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]
: One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with left-over words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
: The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
: Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**− −aefhkntuvx** [*arg* ...]]

- **−a** Mark variables which are modified or created for export.
- **−e** Exit immediately if a command exits with a non-zero exit status.
- **−f** Disable file name generation
- **−h** Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
- **−k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- **−n** Read commands but do not execute them.
- **−t** Exit after reading and executing one command.
- **−u** Treat unset variables as an error when substituting.
- **−v** Print shell input lines as they are read.
- **−x** Print commands and their arguments as they are executed.
- **− −** Do not change any of the flags; useful in setting **$1** to **−**.

Using **+** rather than **−** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given the values of all names are printed.

shift [*n*]

The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test

Evaluate conditional expressions. See *test*(1) for usage and description.

times

Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...

The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]

For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [**−fp**] [*n*]

imposes a size limit of *n*

- **−f** imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.
- **−p** changes the pipe size to *n* (UNIX system/RT only).

If no option is given, **−f** is assumed.

umask [*nnn*]

The user file-creation mask is set to *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

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 flags below are interpreted by the shell on invocation only; Note that unless the **−c** or **−s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

−c *string* If the **−c** flag is present commands are read from *string*.

−s If the **−s** flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.

−i If the **−i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

−r If the **−r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

Rsh Only

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of **$PATH,**
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively 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) in the *UNIX System V Administrator Reference Manual.*
dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), umask(2), wait(2), a.out(4), profile(4), environ(5) in the *UNIX System V Programmer Reference Manual.*

CAVEATS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

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. To have the output of a layer blocked when it is not current, the *stty* option **loblk** may be set within the layer.

The *stty* character **swtch** (set to ^Z if NUL) is used to switch control to *shl* from a layer. *Shl* has its own prompt, **>>>**, to help distinguish it from a layer.

A *layer* is a shell which has been bound to a virtual tty device (**/dev/sxt???**). The virtual device can be manipulated like a real tty device using *stty*(1) and *ioctl*(2). Each layer has its own process group id.

Definitions

A *name* is a sequence of characters delimited by a blank, tab or new-line. Only the first eight characters are significant. The *names* **(1)** through **(7)** cannot be used when creating a layer. They are used by *shl* when no name is supplied. They may be abbreviated to just the digit.

Commands

The following commands may be issued from the *shl* prompt level. Any unique prefix is accepted.

create [*name*]
: Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form *(#)* where # is the last digit of the virtual device bound to the layer. The shell prompt variable **PS1** is set to the name of the layer followed by a space. A maximum of seven layers can be created.

block *name* [*name* ...]
: For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **loblk** within the layer.

delete *name* [*name* ...]
: For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal (see *signal*(2)).

help (or ?)
: Print the syntax of the *shl* commands.

layers [**−l**] [*name* ...]
: For each *name*, list the layer name and its process group. The **−l** option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume [*name*]
: Make the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

toggle Resume the layer that was current before the last current layer.

unblock *name* [*name* ...]
: For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty*

	option **loblk** within the layer.
quit	Exit *shl*. All layers are sent the SIGHUP signal.
name	Make the layer referenced by *name* the current layer.

FILES

/dev/sxt???	Virtual tty devices
$SHELL	Variable containing path name of the shell to use (default is /bin/sh).

SEE ALSO

sh(1), stty(1).
ioctl(2), signal(2) in the *UNIX System V Programmer Reference Manual.*
sxt(7) in the *UNIX System V Administrator Reference Manual.*

NAME

size − print section sizes of common object files

SYNOPSIS

size [−o] [−x] [−V] files

DESCRIPTION

The *size* command produces section size information for each section in the common object files. The size of the text, data and bss (uninitialized data) sections are printed along with the total size of the object file. If an archive file is input to the *size* command the information for all archive members is displayed.

Numbers will be printed in decimal unless either the **−o** or the **−x** option is used, in which case they will be printed in octal or in hexadecimal, respectively.

The **−V** flag will supply the version information on the *size* command.

SEE ALSO

as(1), cc(1), ld(1).
a.out(4), ar(4) in the *UNIX System V Programmer Reference Manual.*

DIAGNOSTICS

size: name: cannot open
    if *name* cannot be read.

size: name: bad magic
    if *name* is not an appropriate common object file.

NAME

sleep — suspend execution for interval

SYNOPSIS

unsigned sleep (seconds)
unsigned seconds;

DESCRIPTION

The current process is suspended from execution for the number of *seconds* specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) Because scheduled wakeups occur at fixed 1-second intervals, (on the second, according to an internal clock) and (2) because any caught signal will terminate the *sleep* following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system. The value returned by *sleep* will be the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested *sleep* time, or premature arousal due to another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling *sleep*. If the *sleep* time exceeds the time till such alarm signal, the process sleeps only until the alarm signal would have occurred. The caller's alarm catch routine is executed just before the *sleep* routine returns. But if the *sleep* time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening *sleep*.

SEE ALSO

alarm(2), pause(2), signal(2).

NAME

sno − SNOBOL interpreter

SYNOPSIS

sno [files]

DESCRIPTION

Sno is a SNOBOL compiler and interpreter (with slight differences). *Sno* obtains input from the concatenation of the named *files* and the standard input. All input through a statement containing the label **end** is considered program and is compiled. The rest is available to **syspit**.

Sno differs from SNOBOL in the following ways:

There are no unanchored searches. To get the same effect:

a ** b	unanchored search for *b*.
a *x* b = x c	unanchored assignment

There is no back referencing.

x = "abc"	
a *x* x	is an unanchored search for **abc**.

Function declaration is done at compile time by the use of the (non-unique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and the use of the name **define** is preempted. There is no provision for automatic variables other than parameters. Examples:

```
define f( )
define f(a, b, c)
```

All labels except **define** (even **end**) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on **end** cannot merely name a label.

If **start** is a label in the program, program execution will start there. If not, execution begins with the first executable statement; **define** is not an executable statement.

There are no built-in functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators / and * must be set off by spaces.

The right side of assignments must be non-empty.

Either ' or " may be used for literal quotes.

The pseudo-variable **sysppt** is not available.

SEE ALSO

awk(1).

NAME

sort − sort and/or merge files

SYNOPSIS

sort [−cmu] [−ooutput] [−ykmem] [−zrecsz] [−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*
The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **−o** and *output.*

−y*kmem*
The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, −y0 is guaranteed to start with minimum memory. By convention, −y (with no argument) starts with maximum memory.

−z*recsz*
The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **−c** or **−m** options, a popular system default size will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

−d "Dictionary" order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

−f Fold lower case letters into upper case.

−i Ignore 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 will be applied to all +*pos1* arguments. Otherwise, the **b** flag 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 flags **bdfinr**. A starting position specified by +*m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*+1st field; +*m*.0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by −*m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; −*m*.1**b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 −2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort −r −o outfile +1.0 −1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

```
sort −r +1.0b −1.1b infile1 infile2
```

Print the password file (*passwd*(4)) sorted by the numeric user ID (the third colon-separated field):

```
sort −t: +2n −3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **−um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort −um +2 −3 infile
```

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

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.

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. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

Spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Under the **−v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated. (Not available on PDP-11.)

Under the **−b** option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Under the **−x** option, every plausible stem is printed with = for each word.

By default, *spell* (like *deroff*(1)) follows chains of included files (**.so** and **.nx** *troff*(1) requests), *unless* the names of such included files begin with **/usr/lib**. Under the **−l** option, *spell* will follow the chains of *all* included files. Under the **−i** option, *spell* will ignore all chains of included files.

Under the *+local_file* option, words found in *local_file* are removed from *spell*'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy−y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

hashmake	Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.
spellin 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; it writes these codes on the standard output.

EXAMPLES

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 & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(1).

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.
The British spelling feature was done by an American.

NAME

spline − interpolate smooth curve

SYNOPSIS

spline [options]

DESCRIPTION

Spline takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed., pp. 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph*(1G).

The following *options* are recognized, each as a separate argument:

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

−k The constant k used in the boundary value computation:
$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$
is set by the next argument (default $k = 0$).

−n Space output points so that approximately n intervals occur between the lower and upper x limits (default $n = 100$).

−p Make output periodic, i.e., match derivatives at ends. First and last input values should normally agree.

−x Next 1 (or 2) arguments are lower (and upper) x limits. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

SEE ALSO

graph(1G).

DIAGNOSTICS

When data is not strictly monotone in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1,000 input points is enforced silently.

NAME

split − split a file into pieces

SYNOPSIS

split [−*n*] [file [name]]

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, **x** is default.

If no input file is given, or if − is given in its stead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

NAME

stat − statistical network useful with graphical commands

SYNOPSIS

node-name [options] [files]

DESCRIPTION

Stat is a collection of command level functions (nodes) that can be interconnected using *sh*(1) to form a statistical network. The nodes reside in **/usr/bin/graf** (see *graphics*(1G)). Data is passed through the network as sequences of numbers (vectors), where a number is of the form:

[sign](digits)(.digits)[**e**[sign]digits]

evaluated in the usual way. Brackets and parentheses surround fields. All fields are optional, but at least one of the fields surrounded by parentheses must be present. Any character input to a node that is not part of a number is taken as a delimiter.

Stat nodes are divided into four classes.

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	[−**c***i*] − round down to next integer
gamma	[−**c***i*] − gamma
list	[−**c***i* **d***string*] − list vector elements delimiter(s)
log	[−**c***i* **b***f*] − logarithm base

mod	[−*ci* **m***f*] − modulus **m**odulus
pair	[−*ci* **F***file* **x***i*] − pair elements **F**ile containing base vector, **x** group size
power	[−*ci* **p***f*] − raise to a power **p**ower
root	[−*ci* **r***f*] − take a root **r**oot
round	[−*ci* **p***i* *si*] − round to nearest integer, .5 rounds to 1 **p**laces after decimal point, **s**ignificant digits
siline	[−*ci* **i***f* **n***i* **s***f*] − generate a line given slope and intercept **i**ntercept, **n**umber of positive integers, **s**lope
sin	[−*ci*] − sine
subset	[−**a***f* **b***f* *ci* **F***file* **i***i* **l***f* **nl** **np** **p***f* *si* *ti*] − generate a subset **a**bove, **b**elow, **F**ile with master vector, **i**nterval, **l**eave, master contains element **n**umbers to **l**eave, master contains element **n**umbers to **p**ick, **p**ick, **s**tart, **t**erminate

Summarizers:

bucket	[−**a***i* *ci* **F***file* **h***f* **i***i* **l***f* **n***i*] − break into buckets **a**verage size, **F**ile containing bucket boundaries, **h**igh, **i**nterval, **l**ow, **n**umber Input data should be sorted
cor	[−**F***file*] − correlation coefficient **F**ile containing base vector
hilo	[− **h l o ox oy**]− find high and low values **h**igh only, **l**ow only, **o**ption form, **o**ption form with **x** prepended, **o**ption form with **y** prepended
lreg	[−**F***file* **i o s**] − linear regression **F**ile containing base vector, **i**ntercept only, **o**ption form for *siline*, **s**lope only
mean	[−**f***f* **n***i* **p***f*] − (trimmed) arithmetic mean **f**raction, **n**umber, **p**ercent
point	[−**f***f* **n***i* **p***f* **s**] − point from empirical cumulative density function **f**raction, **n**umber, **p**ercent, **s**orted input
prod	− internal product
qsort	[−*ci*] − quick sort
rank	− vector rank
total	− sum total
var	− variance

Translators:

bar	[−**a b f g r***i* **w***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] − build a bar chart suppress **a**xes, **b**old, suppress **f**rame, suppress **g**rid, **r**egion, **w**idth in percent, **x** origin, suppress **x**-**a**xis label, **y** origin, suppress **y**-**a**xis label, **y**-axis **l**ower bound, **y**-axis **h**igh bound Data is rounded off to integers.
hist	[−**a b f g r***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] − build a histogram suppress **a**xes, **b**old, suppress **f**rame, suppress **g**rid, **r**egion, **x**

origin, suppress **x-axis** label, **y** origin, suppress **y-axis** label, **y**-axis **l**ower bound, **y**-axis **h**igh 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 **c**ase, label **F**ile, **h**istogram input, **p**lot input, **r**otation, **x**-axis, **u**pper **x**-axis, **y**-axis, **r**ight **y**-axis

pie [**−b o p pn***i* **pp***i* **r***i* **v x***i* **y***i*] − build a pie chart
bold, values **o**utside pie, value as **p**ercentage(:=100), value as **p**ercentage(:=i), draw **p**ercent of **p**ie, **r**egion, no **v**alues, **x** origin, **y** origin
Unlike other nodes, input is lines of the form
[< **i e f c***c* >] value [label]
ignore (do not draw) slice, **e**xplode slice, **f**ill slice, **c**olor slice *c*=(**b**lack, **r**ed, **g**reen, **b**lue)

plot [**−a b c***string* **d f F***file* **g m r***i* **x***f* **xa xi***f* **xh***f* **xl***f* **xn***i* **xt y***f* **ya yi***f* **yh***f* **yl***f* **yn***i* **yt**] − plot a graph
suppress **a**xes, **b**old, plotting **c**haracters, **d**isconnected, suppress **f**rame, **F**ile containing x vector, suppress **g**rid, **m**ark points, **r**egion, **x** origin, suppress **x-a**xis label, **x i**nterval, **x h**igh bound, **x l**ow bound, **n**umber of ticks on **x**-axis, suppress **x**-axis **t**itle, **y** origin, suppress **y-a**xis label, **y i**nterval, **y h**igh bound, **y l**ow bound, **n**umber of ticks on **y**-axis, suppress **y**-axis **t**itle

title [**−b c l***string* **v***string* **u***string*] − title a vector or a GPS
title **b**old, retain **c**ase, **l**ower title, **u**pper title, **v**ector title

Generators:

gas [**−c***i* **i***f* **n***i* **s***f* **t***f*] − generate additive sequence
interval, **n**umber, **s**tart, **t**erminate

prime [**-c***i* **h***i* **l***i* **n***i*] − generate prime numbers
high, **l**ow, **n**umber

rand [**−c***i* **h***f* **l***f* **m***f* **n***i* **s***i*] − generate random sequence
high, **l**ow, **m**ultiplier, **n**umber, **s**eed

RESTRICTIONS

Some nodes have a limit on the size of the input vector.

SEE ALSO

graphics(1G).
gps(4) in the *UNIX System V Programmer Reference Manual.*

NAME

strip – strip symbol and line number information from a common 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 this has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using any of the following options:

−l	Strip line number information only; do not strip any symbol table information.
−x	Do not strip static or external symbol information.
−r	Reset the relocation indexes into the symbol table.
−V	Print the version of the strip command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *strip* will complain and terminate without stripping *file-name* unless the **−r** flag is used.

If the *strip* command is executed on a common archive file (see *ar*(4)) the archive symbol table will be removed. The archive symbol table must be restored by executing the *ar*(1) command with the **s** option before the archive can be link-edited by the *ld*(1) command. *Strip* will instruct the user with appropriate warning messages when this situation arises.

The purpose of this command is to reduce the file storage overhead taken by the object file.

FILES

/usr/tmp/strp??????

SEE ALSO

ar(1), as(1), cc(1), ld(1).
a.out(4), ar(4) in the *UNIX System V Programmer Reference Manual.*

DIAGNOSTICS

strip: name: cannot open
    if *name* cannot be read.

strip: name: bad magic
    if *name* is not an appropriate common object file.

strip: name: relocation entries present; cannot strip
    if *name* contains relocation entries and the **−r** flag is not used, the symbol table information cannot be stripped.

NAME

stty − set the options for a terminal

SYNOPSIS

stty [**−a**] [**−g**] [options]

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the **−a** option, it reports all of the option settings; with the **−g** option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below may be found in *termio*(7) for asynchronous lines, or in *stermio*(7) for synchronous lines in the *UNIX System V Administrator Reference Manual*. Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (**−parenb**)	enable (disable) parity generation and detection.
parodd (**−parodd**)	select odd (even) parity.
cs5 cs6 cs7 cs8	select character size (see *termio*(7)).
0	hang up phone line immediately.
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb	Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)
hupcl (**−hupcl**)	hang up (do not hang up) DATA-PHONE® connection on last close.
hup (**−hup**)	same as **hupcl** (**−hupcl**).
cstopb (**−cstopb**)	use two (one) stop bits per character.
cread (**−cread**)	enable (disable) the receiver.
clocal (**−clocal**)	n assume a line without (with) modem control.
loblk (**−loblk**)	block (do not block) output from a non-current layer.

Input Modes

ignbrk (**−ignbrk**)	ignore (do not ignore) break on input.
brkint (**−brkint**)	signal (do not signal) INTR on break.
ignpar (**−ignpar**)	ignore (do not ignore) parity errors.
parmrk (**−parmrk**)	mark (do not mark) parity errors (see *termio*(7)).
inpck (**−inpck**)	enable (disable) input parity checking.
istrip (**−istrip**)	strip (do not strip) input characters to seven bits.
inlcr (**−inlcr**)	map (do not map) NL to CR on input.
igncr (**−igncr**)	ignore (do not ignore) CR on input.
icrnl (**−icrnl**)	map (do not map) CR to NL on input.
iuclc (**−iuclc**)	map (do not map) 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) or *stermio*(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 will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (**−echok**)	echo (do not echo) NL after KILL character.
lfkc (**−lfkc**)	the same as **echok** (**−echok**); obsolete.
echonl (**−echonl**)	echo (do not echo) NL.
noflsh (**−noflsh**)	disable (enable) flush after INTR, QUIT, or SWTCH.
stwrap (**−stwrap**)	disable (enable) truncation of lines longer than 79 characters on a synchronous line.
stflush (**−stflush**)	enable (disable) flush on a synchronous line after every *write*(2).
stappl (**−stappl**)	use application mode (use line mode) on a synchronous line.

Control Assignments

control-character c	set *control-character* to *c*, where *control-character* is **erase, kill, intr, quit, swtch, eof, ctab, min,** or **time** (**ctab** is used with **−stappl**; see *stermio*(7)), (**min** and **time** are used with **−icanon**; see *termio*(7)). If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "**^d**" is a **CTRL-d**); "**^?**" is interpreted as DEL and "**^−**" is interpreted as undefined.
line *i*	set line discipline to *i* (0 < *i* < 127).

Combination Modes

evenp or **parity**	enable **parenb** and **cs7**.
oddp	enable **parenb**, **cs7**, and **parodd**.
−parity, **−evenp**, or **−oddp**	disable **parenb**, and set **cs8**.
raw (**−raw** or **cooked**)	enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).

nl (**−nl**)	unset (set) **icrnl**, **onlcr**. In addition **−nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (**−lcase**)	set (unset) **xcase**, **iuclc**, and **olcuc**.
LCASE (**−LCASE**)	same as **lcase** (**−lcase**).
tabs (**−tabs** or **tab3**)	preserve (expand to spaces) tabs when printing.
ek	reset ERASE and KILL characters back to normal # and @.
sane	resets all modes to some reasonable values.
term	set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

tabs(1).
ioctl(2) in the *UNIX System V Programmer Reference Manual.*
stermio(7), termio(7) in the *UNIX System V Administrator Reference Manual.*

NAME

su — become super-user or another user

SYNOPSIS

su [−] [name [arg ...]]

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless one is already **root**). If the password is correct, *su* will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry (see *passwd*(4)), or **/bin/sh** if none is specified (see *sh*(1)). To restore normal user ID privileges, type an **EOF** (*cntrl-d*) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like *sh*(1), an *arg* of the form **−c** *string* executes *string* via the shell and an arg of **−r** will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(1). If the first argument to *su* is a −, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is −, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sh**, the user's **.profile** can check *arg0* for **−sh** or **−su** to determine if it was invoked by *login*(1) or *su*(1), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *-program* by both *login*(1) and *su*(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**, type:

```
su - bin -c "command args"
```

FILES

/etc/passwd	system's password file
/etc/profile	system's profile
$HOME/.profile	user's profile
/usr/adm/sulog	log file

SEE ALSO

env(1), login(1), sh(1).
passwd(4), profile(4), environ(5) in the *UNIX System V Programmer Reference Manual.*

NAME

sum − print checksum and block count of a file

SYNOPSIS

sum [**−r**] file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **−r** causes an alternate algorithm to be used in computing the checksum.

SEE ALSO

wc(1).

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

sync − update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync*(2) for details.

SEE ALSO

sync(2) in the *UNIX System V Programmer Reference Manual.*

NAME

tabs − set tabs on a terminal

SYNOPSIS

tabs [tabspec] [**+mn**] [**−T**type]

DESCRIPTION

Tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

Users of GE TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings. The first number in a list of tab settings becomes the *left margin* on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for *tabspec*: "canned," repetitive, arbitrary, and file. If no *tabspec* is given, the default value is **−8**, i.e., UNIX system "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

−code Gives the name of one of a set of "canned" tabs. The legal codes and their meanings are as follows:

−a 1,10,16,36,72
Assembler, IBM S/370, first format

−a2 1,10,16,40,72
Assembler, IBM S/370, second format

−c 1,8,12,16,20,55
COBOL, normal format

−c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:

<:t−c2 m6 s66 d:>

−c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **−c2.** This is the recommended format for COBOL. The appropriate format specification is:

<:t−c3 m6 s66 d:>

−f 1,7,11,15,19,23
FORTRAN

−p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

−s 1,10,55
SNOBOL

−u 1,12,20,44
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

−n A repetitive specification requests tabs at columns 1+*n*, 1+2**n*, etc. Note that such a setting leaves a left margin of *n* columns on TermiNet terminals *only*. Of particular importance is the value **−8**: this represents the UNIX system "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with

the *nroff* **−h** option for high-speed output. Another special case is the value **−0**, implying no tabs at all.

n1,n2,... The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

−−*file* If the name of a file is given, *tabs* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **−8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:

```
tabs −− file; pr file
```

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

−T*type* *Tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in *term*(5). If no **−T** flag is supplied, *tabs* searches for the **$TERM** value in the *environment* (see *environ*(5)). If no *type* can be found, *tabs* tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

DIAGNOSTICS

illegal tabs	when arbitrary tabs are ordered incorrectly.
illegal increment	when a zero or missing increment is found in an arbitrary specification.
unknown tab code	when a "canned" code cannot be found.
can't open	if −−*file* option used, and file can't be opened.
file indirection	if −−*file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

SEE ALSO

pr(1).
environ(5), term(5) in the *UNIX System V Programmer Reference Manual.*

BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.
It is generally impossible to usefully change the left margin without also setting tabs.
Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

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 will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail −f fred
```

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

```
tail −15cf fred
```

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

SEE ALSO

dd(1).

BUGS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

NAME

tar — tape file archiver

SYNOPSIS

tar [key] [files]

DESCRIPTION

Tar saves and restores files on magnetic tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named *files* are written on the end of the tape. The **c** function implies this function.

x The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. If a named file on tape does not exist on the system, the file is created with the same mode as the one on tape except that the set-user-ID and set-group-ID bits are not set unless you are super-user. If the files exist, their modes are not changed except for the bits described above. The owner, group, and modification time are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

t The names of all the files on the tape are listed.

u The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape.

c Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This command implies the **r** function.

The following characters may be used in addition to the letter that selects the desired function:

#s Where # is a tape drive number (**0,...,7**), and **s** is the density (**l** - low (800 bpi), **m** - medium (1600 bpi), or **h** - high (6250 bpi)). This modifier selects the drive on which the tape is mounted. The default is **0m**.

v Normally, *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w Causes *tar* to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means "no".

f Causes *tar* to use the next argument as the name of the archive instead of **/dev/mt/??**. If the name of the file is **—**, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *Tar* can also be used to move hierarchies with the command:

```
cd fromdir; tar cf — . | (cd todir; tar xf —)
```

b Causes *tar* to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes (key letters **x** and **t**).

l Tells *tar* to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.

m Tells *tar* not to restore the modification times. The modification time of the file will be the time of extraction.

o Causes extracted files to take on the user and group identifier of the user running the program rather than those on the tape.

FILES

/dev/mt/*
/tmp/tar*

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

BUGS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
The **b** option should not be used with archives that are going to be updated. The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.
The current limit on file-name length is 100 characters.
Note that **tar c0m** is not the same as **tar cm0**.

NAME

tee − pipe fitting

SYNOPSIS

tee [**−i**] [**−a**] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. The **−i** option ignores interrupts; the **−a** option causes the output to be appended to the *files* rather than overwriting them.

NAME

test − condition evaluation command

SYNOPSIS

test expr
[expr]

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

−r *file*	true if *file* exists and is readable.
−w *file*	true if *file* exists and is writable.
−x *file*	true if *file* exists and is executable.
−f *file*	true if *file* exists and is a regular file.
−d *file*	true if *file* exists and is a directory.
−c *file*	true if *file* exists and is a character special file.
−b *file*	true if *file* exists and is a block special file.
−p *file*	true if *file* exists and is a named pipe (fifo).
−u *file*	true if *file* exists and its set-user-ID bit is set.
−g *file*	true if *file* exists and its set-group-ID bit is set.
−k *file*	true if *file* exists and its sticky bit is set.
−s *file*	true if *file* exists and has a size greater than zero.
−t [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
−z *s1*	true if the length of string *s1* is zero.
−n *s1*	true if the length of the string *s1* is non-zero.
s1 **=** *s2*	true if strings *s1* and *s2* are identical.
s1 **!=** *s2*	true if strings *s1* and *s2* are *not* identical.
s1	true if *s1* is *not* the null string.
n1 **−eq** *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, and **−le** may be used in place of **−eq**.

These primaries may be combined with the following operators:

!	unary negation operator.
−a	binary *and* operator.
−o	binary *or* operator (**−a** has higher precedence than **−o**).
(expr)	parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

SEE ALSO

find(1), sh(1).

WARNING

In the second form of the command (i.e., the one that uses [], rather than the word *test*), the square brackets must be delimited by blanks.
Some UNIX systems do not recognize the second form of the command.

NAME

time − time a command

SYNOPSIS

time command

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on standard error.

SEE ALSO

timex(1).
times(2) in the *UNIX System V Programmer Reference Manual.*

CAVEATS

When *time* is used on a 3B 20A dual computer system the sum of system and user time could be greater than real time. This is the result when *command* is a multi-threaded task running on a 3B 20A computer system with both processors active.

NAME

timex − time a command; report process data and system activity

SYNOPSIS

timex [options] command

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on standard error.

Options are:

−p List process accounting records for *command* and all its children. Suboptions **f, h, k, m, r,** and **t** modify the data items reported, as defined in *acctcom*(1). The number of blocks read or written and the number of characters transferred are always reported.

−o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

−s Report total system activity (not just that due to *command)* that occurred during the execution interval of *command.* All the data items listed in *sar*(1) are reported.

SEE ALSO

acctcom(1), sar(1).

CAVEATS

When *timex* is used on a 3B 20A dual computer system the sum of system and user time could be greater than real time. This is the result when *command* is a multi-threaded task runing on a 3B 20A computer system with both processors active.

WARNING

Process records associated with *command* are selected from the accounting file **/usr/adm/pacct** by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

EXAMPLES

A simple example:

```
timex −ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
timex −opskmt sh

        session commands
EOT
```

NAME

toc − graphical table of contents routines

SYNOPSIS

dtoc [directory]
ttoc mm-file
vtoc [−**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 .). The list has one entry per directory. The entry fields from left to right are level number, directory name, and the number of ordinary readable files in the directory. *Dtoc* is useful in making a visual display of all or parts of a file system. The following will make a visual display of all the readable directories under /:

dtoc / | vtoc | td

ttoc Output is the table of contents generated by the .TC macro of *mm*(1) translated to TTOC format. The input is assumed to be an *mm* file that uses the .H family of macros for section headers. If no *file* is given, the standard input is assumed.

vtoc *Vtoc* produces a GPS describing a hierarchy chart from a TTOC. The output drawing consists of boxes containing text connected in a tree structure. If no *file* is given, the standard input is assumed. Each TTOC entry describes one box and has the form:

id [*line-weight*,*line-style*] "*text*" [*mark*]

where:

id is an alternating sequence of numbers and dots. The *id* specifies the position of the entry in the hierarchy. The *id* **0.** is the root of the tree.

line-weight is either:

n, normal-weight; or
m, medium-weight; or
b, bold-weight.

line-style is either:

so, solid-line;
do, dotted-line;
dd, dot-dash line;
da, dashed-line; or
ld, long-dashed

text is a character string surrounded by quotes. The characters between the quotes become the contents of the box. To include a quote within a box it must be escaped (\").

mark is a character string (surrounded by quotes if it contains spaces), with included dots being escaped. The string is put above the top right corner of the box. To include either a quote or a dot within a *mark* it must be escaped.

Entry example: 1.1 b,da "ABC" DEF
Entries may span more than one line by escaping the new-line (**\new-line**).

Comments are surrounded by the **/*,*/** pair. They may appear anywhere in a TTOC.

Options:

c	Use text as entered (default is all upper case).
d	Connect the boxes with diagonal lines.
h*n*	Horizontal interbox space is *n*% of box width.
i	Suppress the box *id*.
m	Suppress the box *mark*.
s	Do not compact boxes horizontally.
v*n*	Vertical interbox space is *n*% of box height.

SEE ALSO

graphics(1G).
gps(4) in the *UNIX System V Programmer Reference Manual.*

NAME

touch − update access and modification times of a file

SYNOPSIS

touch [**−amc**] [mmddhhmm[yy]] files

DESCRIPTION

Touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see *date*(1)) the current time is used. The **−a** and **−m** options cause touch to update only the access or modification times respectively (default is **−am**). The **−c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1).
utime(2) in the *UNIX System V Programmer Reference Manual.*

NAME

tplot − graphics filters

SYNOPSIS

tplot [**−T**terminal [**−e** raster]]

DESCRIPTION

These commands read plotting instructions (see *plot*(4)) from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter **$TERM** (see *environ*(5)) is used. Known *terminal*s are:

300	DASI 300.
300S	DASI 300s.
450	DASI 450.
4014	TEKTRONIX 4014.
ver	Versatec D1200A. This version of *plot* places a scan-converted image in **/usr/tmp/raster$$** and sends the result directly to the plotter device, rather than to the standard output. The **−e** option causes a previously scan-converted file *raster* to be sent to the plotter.

FILES

/usr/lib/t300
/usr/lib/t300s
/usr/lib/t450
/usr/lib/t4014
/usr/lib/vplot
/usr/tmp/raster$$

SEE ALSO

plot(3X), plot(4), term(5) in the *UNIX System V Programmer Reference Manual.*

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 flag 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) in the *UNIX System V Programmer Reference Manual.*

NAME

tr − translate characters

SYNOPSIS

tr [**−cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **−cds** may be used:

−c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.

−d Deletes all input characters in *string1*.

−s Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a−z] Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.

[a*n] Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of 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) in the *UNIX System V Programmer Reference Manual.*

BUGS

Will not handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

NAME

true, false — provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true
do
        command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tsort − topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty − get the name of the terminal

SYNOPSIS

tty [**−l**] [**−s**]

DESCRIPTION

Tty prints the path name of the user's terminal. The **−l** option prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line. The **−s** option inhibits printing of the terminal path name, allowing one to test just the exit code.

EXIT CODES

2	if invalid options were specified,
0	if standard input is a terminal,
1	otherwise.

DIAGNOSTICS

"not on an active synchronous line" if the standard input is not a synchronous terminal and **−l** is specified.
"not a tty" if the standard input is not a terminal and **−s** is not specified.

NAME

umask − set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

Umask is recognized and executed by the shell.

SEE ALSO

chmod(1), sh(1).
chmod(2), creat(2), umask(2) in the *UNIX System V Programmer Reference Manual.*

NAME

uname — print name of current UNIX system

SYNOPSIS

uname [**−snrvma**]

DESCRIPTION

Uname prints the current system name of the UNIX system on the standard output file. It is mainly useful to determine which system one is using. The options cause selected information returned by *uname*(2) to be printed:

−s	print the system name (default).
−n	print the nodename (the nodename may be a name that the system is known by to a communications network).
−r	print the operating system release.
−v	print the operating system version.
−m	print the machine hardware name.
−a	print all the above information.

SEE ALSO

uname(2) in the *UNIX System V Programmer Reference Manual.*

NAME

unget – undo a previous get of an SCCS file

SYNOPSIS

unget [–**r**SID] [–**s**] [–**n**] files

DESCRIPTION

Unget undoes the effect of a **get –e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

–**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 keyletter 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 omitted on the command line.
–**s**	Suppresses the printout, on the standard output, of the intended delta's *SID*.
–**n**	Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO

delta(1), get(1), help(1), sact(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq − report repeated lines in a file

SYNOPSIS

uniq [**−udc** [+n] [−n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **−u** flag is used, just the lines that are not repeated in the original file are output. The **−d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **−u** and **−d** mode outputs.

The **−c** option supersedes **−u** and **−d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

−*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), sort(1).

NAME

units − conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e−01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e−01
```

Units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as **g**,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	astronomical unit.

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

/usr/lib/unittab

NAME

uucp, uulog, uuname — UNIX system to UNIX system copy

SYNOPSIS

uucp [options] source-files destination-file

uulog [options]

uuname [**−l**] [**−v**]

DESCRIPTION

Uucp

Uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names which *uucp* knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, and only to a destination in PUBDIR (see below). Care should be taken to insure that intermediate nodes in the route are willing to foward information.

The shell metacharacters **?**, ***** and **[...]** appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

(1) a full path name;

(2) a path name preceded by ~*user* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by ~/*user* where *user* is a login name on the specified system and is replaced by that user's directory under PUBDIR;

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

The following options are interpreted by *uucp*:

−d Make all necessary directories for the file copy (default).

−f Do not make intermediate directories for the file copy.

−c Use the source file when copying out rather than copying the file to the spool directory (default).

−C Copy the source file to the spool directory.

−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.

−e*sys* Send the *uucp* command to system *sys* to be executed there. (Note: this will only be successful if the remote machine allows the *uucp* command to be executed by **/usr/lib/uucp/uuxqt**.)

−r Queue job but do not start the file transfer process. By default a file transfer process is started each time uucp is evoked.

−j Control writing of the *uucp* job number to standard output (see below).

Uucp associates a job number with each request. This job number can be used by *uustat* to obtain status or terminate the job.

The environment variable **JOBNO** and the **-j** option 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 will not be listed (default). If *uucp* is then invoked with the **-j** option, the job number will be listed. If the environment variable **JOBNO** is set to **ON** and is exported, a job number will be written to standard output each time uucp is invoked. In this case, the **-j** option will supress output of the job number.

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 will be printed.

−u*user* Print information about work done for the specified, *user*. If *user* is not specified then logging information for all users will be printed.

Uuname

Uuname lists the uucp names of known systems. The **−l** option returns the local system name. The **−v** option will print additional information about each system. A description will be printed for each system that has a line of information in **/usr/lib/uucp/ADMIN**. The format of ADMIN is: *sysname* tab *description* tab.

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), uux(1C).
chmod(2) in the *UNIX System V Programmer Reference Manual.*

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 ~).

NOTES

In order to send files that begin with a dot (e.g., .profile) the files must by qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

Uucp will not generate a job number for a strictly local transaction.

BUGS

All files received by *uucp* will be owned by *uucp*.
The **−m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters **? * [...]** will not activate the **−m** option.
The **−m** option will not work if all transactions are local or if **uucp** is executed remotely via the **−e** option.
The **−n** option will function 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.

NAME

uustat − uucp status inquiry and job control

SYNOPSIS

uustat [options]

DESCRIPTION

Uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. The following *options* are recognized:

−**j***jobn*	Report the status of the *uucp* request *jobn*. If **all** is used for *jobn*, the status of all *uucp* requests is reported. An argument must be supplied; otherwise, the usage message will be printed and the request will fail.
−**k***jobn*	Kill the *uucp* request whose job number is *jobn*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.
−**r***jobn*	Rejuvenate *jobn*. That is, *jobn* is touched so that its modification time is set to the current time. This prevents *uuclean* from deleting the job until the jobs modification time reaches the limit imposed by *uuclean*.
−**c***hour*	Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the super-user.
−**u***user*	Report the status of all *uucp* requests issued by *user*.
−**s***sys*	Report the status of all *uucp* requests which communicate with remote system *sys*.
−**o***hour*	Report the status of all *uucp* requests which are older than *hour* hours.
−**y***hour*	Report the status of all *uucp* requests which are younger than *hour* hours.
−**m***mch*	Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local *uucp* are provided.
−**M***mch*	This is the same as the −*m* option except that two times are printed. The time that the last status was obtained and the time that the last successful transfer to that system occurred.
−O	Report the *uucp* status using the octal status codes listed below. If this option is not specified, the verbose description is printed with each *uucp* request.
−q	List the number of jobs and other control files queued for each machine and the time of the oldest and youngest file queued for each machine. If a lock file exists for that system, its date of creation is listed.

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user. Note that only one of the options **−j**, **−m**, **−k**, **−c**, **−r**, can be used with the rest of the other options.

For example, the command:

```
uustat −uhdc −smhtsa −y72
```

will print the status of all *uucp* requests that were issued by user *hdc* to communicate with system *mhtsa* within the last 72 hours. The meanings of the job request status are:

job-number user remote-system command-time status-time status where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
000001	the copy failed, but the reason cannot be determined
000002	permission to access local file is denied
000004	permission to access remote file is denied
000010	bad *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).

NAME

uuto, uupick − public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [options] source-files destination
uupick [−s system]

DESCRIPTION

Uuto sends *source-files* to *destination*. *Uuto* uses the *uucp*(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *Logname* is the login name of someone on the specified system.

Two *options* are available:

−p	Copy the source file into the spool directory before transmission.
−m	Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by *mail*(1) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from *system*: [file *file-name*] [dir *dirname*] **?**

Uupick then reads a line from the standard input to determine the disposition of the file:

<new-line>	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir* (current directory is default).
a [*dir*]	Same as **m** except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

Uupick invoked with the **−s***system* option will only search the PUBDIR for files sent from *system*.

FILES

PUBDIR/usr/spool/uucppublic	public directory

NOTES

In order to send files that begin with a dot (e.g., .profile) the files must by qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

SEE ALSO

mail(1), uucp(1C), uustat(1C), uux(1C).
uuclean(1M) in the *UNIX System V Administrator Reference Manual.*

NAME

uux − UNIX-to-UNIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail*(1)) via *uux*.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

(1) a full path name;

(2) a path name preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

As an example, the command

```
uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff"
```

will get the **f1** files from the "usg" and "pwba" machines, execute a *diff* command and put the results in **f1.diff** in the local directory.

Any special shell characters such as **<>;|** should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

Uux will attempt to get all files to the execution system. For files which are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!uucp b!/usr/file \(c!/usr/file\)
```

will send a *uucp* command to system "a" to get **/usr/file** from system "b" and send it to system "c".

Uux will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine. 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. See the *UUCP Administrator Manual* in the *UNIX System V Administrator Guide*.

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

−	The standard input to *uux* is made the standard input to the *command-string*.
−n	Send no notification to user.

−m*file* Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

−j Control writing of the *uucp* job number to standard output.

Uux associates a job number with each request. This job number can be used by *uustat* to obtain status 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 will not be listed (default). If *uuco* is then invoked with the **−j** option, the job number will be listed. If the environment variable **JOBNO** is set to **ON** and is exported, a job number will be written to standard output each time *uux* is invoked. In this case, the **−j** option will suppress output of the job number.

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).

BUGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.
Only the first six characters of the *system-name* are significant. Any excess characters are ignored.

NAME

val − validate SCCS file

SYNOPSIS

val −
val [−**s**] [−**r**SID] [−**m**name] [−**y**type] files

DESCRIPTION

Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a −, and named files.

Val has a special argument, −, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

−**s**	The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.
−**r***SID*	The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (e. g., **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e. g., **r**1.0 or **r**1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.
−**m***name*	The argument value *name* is compared with the SCCS %M% keyword in *file*.
−**y***type*	The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, −**y** mismatch;
bit 7 = %M%, −**m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned − a logical **OR** of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

NAME

vc − version control

SYNOPSIS

vc [−a] [−t] [−cchar] [−s] [keyword=value ... keyword=value]

DESCRIPTION

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the **−t** keyletter (see below). The default control character is colon (:), except as modified by the **−c** keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with *ed*(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The **−a** keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments

−a	Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.
−t	All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.
−c*char*	Specifies a control character to be used in place of :.
−s	Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements

:dcl keyword[, ..., keyword]

Used to declare keywords. All keywords must be declared.

:asg keyword=value

Used to assign values to keywords. An **asg** statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous **asg**'s for that keyword. Keywords declared, but not assigned values have null values.

:if condition

.
.
.

:end

Used to skip lines of the standard input. If the condition is true all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.
The syntax of a condition is:

```
<cond>        ::= [ "not" ] <or>
<or>          ::= <and> | <and> "|" <or>
<and>         ::= <exp> | <exp> "&" <and>
<exp>         ::= "(" <or> ")" | <value> <op> <value>
<op>          ::= "=" | "!=" | "<" | ">"
<value>       ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

```
= != > <     all of equal precedence
&
|
```

Parentheses may be used to alter the order of precedence.
Values must be separated from operators or parentheses by at least one blank or tab.

::text
Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the **−a** keyletter.

:on

:off
Turn on or off keyword replacement on all lines.

:ctl char
Change the control character to char.

:msg message
Prints the given message on the diagnostic output.

:err message

Prints the given message followed by:

ERROR: err statement on line ... (915)

on the diagnostic output. *Vc* halts execution, and returns an exit code of 1.

SEE ALSO

ed(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

EXIT CODES

0 — normal
1 — any error

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 an 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 *Vi Quick Reference* card, the *Introduction to Display Editing with Vi* and the *Ex Reference Manual* provide full details on using *vi*.

INVOCATION

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 will be printed.
−l	**LISP** mode; indents 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	Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.
−R	Read only mode; the **readonly** flag is set, preventing accidental overwriting of the file.
+*command*	The specified *ex* command is interpreted before editing begins.

The *name* argument indicates files to be edited.

The *view* invocation is the same as *vi* except that the **readonly** flag is set.

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.

"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*
ea*s*ESC	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	nth line after *pat*
?pat?−n	nth 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

nsert 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

AUTHOR

Vi and *ex* were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

ex (1).

Vi Quick Reference Card.
An Introduction to Display Editing with Vi, and *Ex Reference Manual* in the *UNIX System Documentation Workbench.*

CAVEATS AND BUGS

The commands which are not supported are detailed in "An Introduction to Display Editing with Vi". The most notable commands which are missing are the macro and abbreviation facilities, and the *vedit* invocation.

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.

There should be an interactive *help* facility and a tutorial suited for beginners.

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).
wait(2) in the *UNIX System V Programmer Reference Manual.*

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

NAME

wc − word count

SYNOPSIS

wc [**−lwc**] [names]

DESCRIPTION

Wc counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **−lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

NAME

what − identify SCCS files

SYNOPSIS

what [**−s**] files

DESCRIPTION

What searches the given files for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

What is intended to be used in conjunction with the command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

−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.

BUGS

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

who − who is on the system

SYNOPSIS

who [**−uTHlpdbrtasq**] [file]

who am i

who am I

DESCRIPTION

Who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX system user. It examines the **/etc/utmp** file to obtain its information. If *file* is given, that file is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

Who with the **am i** or **am I** option identifies the invoking user.

Except for the default **−s** option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

−u This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (**.**) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in **/etc/inittab** (see *inittab*(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.

−T This option is the same as the **−u** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A **+** appears if the terminal is writable by anyone; a **−** appears if it is not. **Root** can write to all lines having a **+** or a **−** in the *state* field. If a bad line is encountered, a **?** is printed.

−l This option lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

−H This option will print column headings above the regular output.

−q This is a quick *who,* displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

−p This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from **/etc/inittab** that spawned this process. See *inittab*(4).

−d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait*(2)), of the dead process. This can be useful in determining why a process terminated.

−b This option indicates the time and date of the last reboot.

−r This option indicates the current *run-level* of the *init* process.

−t This option indicates the last change to the system clock (via the *date*(1) command) by **root**. See *su*(1).

−a This option processes **/etc/utmp** or the named *file* with all options turned on.

−s This option is the default and lists only the *name*, *line*, and *time* fields.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), login(1), mesg(1), su(1).
wait(2), inittab(4), utmp(4) in the *UNIX System V Programmer Reference Manual.*
init(1M) in the *UNIX System V Administrator Reference Manual.*

NAME

write — write to another user

SYNOPSIS

write user [line]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *yourname* **(tty??)** [*date*]...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n". At that point *write* writes **EOT** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., **tty00**); otherwise, the first writable instance of the user found in **/etc/utmp** is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal

Permission to write may be denied or granted by use of the *mesg(1)* command. Writing to others is normally allowed by default. Certain commands, in particular *nroff*(1) and *pr*(1) disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., **(o)** for "over") so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute !

SEE ALSO

mail(1), mesg(1), nroff(1), pr(1), sh(1), who(1).

DIAGNOSTICS

"*user is not logged on*" if the person you are trying to *write* to is not logged on.
"*Permission denied*" if the person you are trying to *write* to denies that permission (with *mesg*).
"*Warning: cannot respond, set mesg -y*" if your terminal is set to *mesg n* and the recipient cannot respond to you.
"*Can no longer write to user*" if the recipient has denied permission (*mesg n*) after you had started writing.

NAME

xargs − construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

Xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

Command, which may be a shell file, is searched for, using one's **$PATH**. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **−i** flag). Flags **−i**, **−l**, and **−n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **−l** vs. **−n**), the last flag has precedence. *Flag* values are:

−l*number* — *Command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **−x** is forced.

−i*replstr* — Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **−x** is also forced. **{}** is assumed for *replstr* if not specified.

−n*number* — Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **−x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

−t	Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.
−p	Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**−t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.
−x	Causes *xargs* to terminate if any argument list would be greater than *size* characters; **−x** is forced by the options **−i** and **−l**. When neither of the options **−i**, **−l**, or **−n** are coded, the total length of all arguments must be within the *size* limit.
−s*size*	The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **−s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.
−e*eofstr*	*Eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical **EOF** string if **−e** is not coded. The value **−e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). *Xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

Xargs will terminate if either it receives a return code of **−1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh*(1)) with an appropriate value to avoid accidentally returning with **−1**.

EXAMPLES

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs −i −t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

```
1.  ls | xargs −p −l ar r arch
2.  ls | xargs −p −l | xargs ar r arch
```

The following will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs −n2 diff
```

SEE ALSO

sh(1).

DIAGNOSTICS

Self-explanatory.

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; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *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*.

If the **−v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **−d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

If the **−l** flag is given, the code produced in **y.tab.c** will *not* contain any **#line** constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when *yacc*'s **−t** option is used, this debugging code will be compiled by default. Independent of whether the **−t** option was used, the runtime debugging code is under the control of **YYDEBUG**, a pre-processor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

FILES

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).
malloc(3X) in the *UNIX System V Programmer Reference Manual.*

YACC−Yet Another Compiler Compiler in the *UNIX System V Support Tools Guide*.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

READER COMMENT FORM

UNIX SYSTEM V
User Reference

Altos Computer Systems
2641 Orchard Parkway
San Jose, CA 95134

This document has been prepared for use with your Altos Computer System. Should you find an error or problem in this manual, please write it down (noting page number), and return this form to the ALTOS PUBLICATIONS DEPARTMENT.

System Model Number____________________

Serial Number________________

Document Title____

Revision Number 69Ø-15834-ØØ1 Date______________________

Name____

Company Name____

Address____

ALTOS
COMPUTER SYSTEMS®

Printed in U.S.A.
P/N 690-15834-001

2641 Orchard Park Way, San Jose, California 95134
(408) 946-6700, Telex 470642 Alto UI

July 1985