NAME

prs - print an SCCS file

SYNOPSIS

prs [**-d**[dataspec]] [**-r**[SID]] [**-e**] [**-l**] [**-a**] file ...

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(5)) 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 pathname 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. Again, non-SCCS files and unreadable files are silently ignored.

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

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

-d[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see **DATA KEYWORDS**) interspersed with optional user supplied text.

-r[*SID*] Used to specify the *SCCS IDentification* (**SID**) string of a delta for which information is desired. If no **SID** is specified, the **SID** of the most recently created delta is assumed.

-e Requests information for all deltas created *earlier* than and including the delta designated via the **-r** keyletter.

-l Requests information for all deltas created *later* than and including the delta designated via the **-r** keyletter.

-a Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1S)) 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*(5)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by \t and carriage return/new line is specified by \n.

TABLE 1. SCCS FILES DATA KEYWORDS

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	see below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Pgmmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq no.	"	nnnn	S
:DI:	Seq no. of deltas included, excluded, 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

NAME

prt - print SCCS file

SYNOPSIS

prt [**-d**] [**-s**] [**-a**] [**-i**] [**-u**] [**-f**] [**-t**] [**-b**] [**-e**] [**-y**[SID]] [**-c**[cutoff]] [**-r**[rev-cutoff]] file ...

DESCRIPTION

Prt prints part or all of an SCCS file in a useful format. If a directory is named, *prt* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname 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, but the effects of any keyletter argument apply independently to each named file.

-d This keyletter normally causes the printing of delta table entries of the **D** type.

-s Causes only the first line of the delta table entries to be printed; that is, only up to the statistics. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

-a Causes those types of deltas normally not printed by the **d** keyletter to be printed. These are types **R** (removed). This keyletter is effective only if the **d** keyletter is also specified (or assumed).

-i Causes the printing of the serial numbers of those deltas included, excluded, and ignored. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

The following format is used to print those portions of the SCCS file as specified by the above keyletters. The printing of each delta table entry is preceded by a newline character.

a) Type of delta (**D** or **R**).
b) Space.
c) **SCCS** identification string (**SID**).
d) Tab.
e) Date and time of creation.
(in the form YY/MM/DD HH:MM:SS)
f) Tab.
g) Creator.
h) Tab.
i) Serial number.
j) Tab.
k) Predecessor delta's serial number.
l) Tab.
m) Statistics.
(in the form inserted/deleted/unchanged)
n) Newline.
o) "Included:*tab*", followed by **SID**'s of deltas included, followed by newline (only if there were any such deltas and if **i** keyletter was supplied).
p) "Excluded:*tab*", followed by **SID**'s of deltas excluded, followed by newline (see note above).

q) "Ignored:*tab*", followed by SID's of deltas ignored, followed by newline (see note above).

r) "MRs:*tab*", followed by MR numbers related to the delta, followed by newline (only if any MR numbers were supplied).

s) Lines of comments (delta commentary), followed by newline (if any were supplied).

-u Causes the printing of the login-names and/or numerical group IDs of those users allowed to make deltas.

-f Causes the printing of the flags of the named file.

-t Causes the printing of the descriptive text contained in the file.

-b Causes the printing of the body of the SCCS file.

-e This keyletter implies the **d**, **i**, **u**, **f**, and **t** keyletters and is provided for convenience.

-y[*SID*] This keyletter will cause the printing of the delta table entries to stop when the delta just printed has the specified SID. If no delta in the table has the specified SID, the entire table is printed. If no SID is specified, the first delta in the delta table is printed. This keyletter will cause the entire delta table entry for each delta to be printed as a single line (the newlines in the normal multi-line format of the **d** keyletter are replaced by blanks) preceded by the name of the SCCS file being processed, followed by a :, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

-c[*cutoff*] This keyletter will cause the printing of the delta table entries to stop if the delta about to be printed is older than the specified cutoff date-time (see *get*(1S) for the format of date-time). If no date-time is supplied, the epoch 0000 GMT Jan. 1, 1970 is used. As with the **y** keyletter, this keyletter will cause the entire delta table entry to be printed as a single line and to be preceded by the name of the SCCS file being processed, followed by a :, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

-r[*rev-cutoff*] This keyletter will cause the printing of the delta table entries to begin when the delta about to be printed is older than or equal to the specified cutoff date-time (see *get*(1S) for the format of date-time). If no date-time is supplied, the epoch 0000 GMT Jan. 1, 1970 is used. (In this case, nothing will be printed). As with the **y** keyletter, this keyletter will cause the entire delta table entry to be printed as a single line and to be preceded by the name of the SCCS file being processed, followed by a :, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

If any keyletter but **y**, **c**, or **r** is supplied, the name of the file being processed (preceded by one newline and followed by two newlines) is printed before its contents.

If none of the **u** , **f** , **t** , or **b** keyletters is supplied, the **d** keyletter is assumed.

Note that the **s** and **i** keyletters, and the **c** and **r** keyletters are mutually exclusive; therefore, they may not be specified together on the same *prt* command.

The form of the delta table as produced by the **y**, **c**, and **r** keyletters makes it easy to sort multiple delta tables by time order. For example, the following will print the delta tables of all SCCS files in directory *sccs* in *rev*erse chronological order:

```
prt –c sccs | grep . | sort '–rt<tab>' +2 –3
```

When both the **y** and **c** or the **y** and **r** keyletters are supplied, *prt* will stop printing when the first of the two conditions is met.

The *reform*(1S) command can be used to truncate long lines.

See *admin*(1S), *sccsfile*(5), and *Source Code Control System User's Guide* for more information about the meaning of the output of *prt*.

SEE ALSO

admin(1S), delta(1S), get(1S), help(1S), what(1S), sccsfile(5)
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1S) for explanations.

NAME

ps − process status

SYNOPSIS

ps [**−aklx**] [name]

DESCRIPTION

Ps prints certain indicia about active processes. The **a** flag asks for information about all processes with process groups (ordinarily only one's own processes are displayed); **x** asks even about processes with no process group; **l** asks for a long listing. Ordinarily only the line number (if not one's own), the process number, its parent's process number, its process group, and an approximation to the command line are given. If the **k** flag is specified, the file unix-core is used in place of /dev/mem. This is used for post-mortem system debugging. Name is the name of a file containing the system namelist for the running system if it is not in /unix.

The long listing is columnar and contains

The line number (00 through 99) of the control terminal of the process.

A number encoding the flags associated with a process. Any combination of the following,

01 process swapped in.

02 The UNIX Scheduler

04 Locked in core

30 Tracing

100 Sleeping

A letter encoding the state of a process.

S − Sleeping

W − Waiting

R − Running

I − Idling (unused)

Z − Zombie − process exited, parent not yet notified.

T − Traced

A number related in some unknown way to the scheduling heuristic.

The priority of the process; high numbers mean low priority.

The nice of the process; high numbers mean low priority.

The process unique number (as in certain cults it is possible to kill a process if you know its true name).

The process number of the parent of the process.

The process group of the process.

An entry that tells the core address in the system of the event which the process is waiting for; if blank, the process is running.

The last column is the file name of the process, plus a few of the arguments, if any, which were passed.

Ps makes an educated guess as to the file name and arguments given when the process was created by examining core memory or the swap area. The method is inherently somewhat unreliable and in any event a process in entitled to destroy this information, so the names cannot be counted on too much. *Ps* also assumes a swap

device but is intelligent enough to figure it out for itself.

FILES

/unix	system namelist
/dev/mem	core memory
/dev/swapdev	swap device
unixcore	optional mem file

SEE ALSO

kill(1), sps(1)

NAME

ptx — permuted index

SYNOPSIS

ptx [option] ... [input [output]]

DESCRIPTION

Ptx generates a permuted index to file *input* on file *output* (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 the page. *Ptx* produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx may be an *nroff* or *troff*(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head,* at least one of which is always an 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; the default line length is 100 characters.

−w *n* Use the next argument, *n,* as the width of the output line. The default line length is 72 characters.

−g *n* Use the next argument, *n,* as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

−o only Use as keywords only the words given in the *only* file.

−i ignore Do not use as keywords any words given in the *ignore* file. If the **−i** and **−o** options are missing, use **/usr/lib/eign** as the *ignore* file.

−b break Use the characters in the *break* file to separate words. In any case, tab, new-line, and space characters are always used as break characters.

−r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx.*

FILES

/bin/sort
/usr/lib/eign

BUGS

Line length counts do not account for overstriking or proportional spacing.
Lines that contain tilde (~) are botched, as *ptx* uses that character internally.

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

SEE ALSO

cd(1)

DIAGNOSTICS

'Cannot open ..' and 'Read error in ..' indicate possible file system trouble and should be referred to a UNIX programming counselor. One possible problem is a file system mounted on a node which is not a directory.

NAME

quot − summarize file system ownership

SYNOPSIS

quot [option] ... [filesystem]

DESCRIPTION

Quot prints the number of inodes and blocks in the named *filesystem* currently owned by each user and the total and free inode counts. If no *filesystem* is named, a default name is assumed. The following options are available:

- **−n** Cause the pipeline **ncheck filesystem | sort +0n | quot −n filesystem** to produce a list of all files and their owners.
- **−i** Print only the total and free inode counts.
- **−f** Causes the output list to be sorted on inode usage.

FILES

/etc/passwd to get user names

SEE ALSO

ls(1), du(1)

BUGS

Holes in files are counted as if they actually occupied space.

NAME

ratfor − rational FORTRAN dialect

SYNOPSIS

ratfor [option ...] [filename ...]

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)

returns expression to caller from function

define: define name replacement

include:

include filename

Ratfor is best used with *f77*(1).

SEE ALSO

f77(1)

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

NAME

readl – read one line

SYNOPSIS

readl

DESCRIPTION

Readl copies one line (up to newline) from the standard input and writes it on the standard output. It is often used within shell files to read from the user's terminal.

SEE ALSO

sh(1), read(2)

BUGS

Readl truncates after 255 characters.

NAME

readnews - read news articles

SYNOPSIS

readnews [**-a** [date]] [**-n** newsgroups] [**-t** titles] [**-lprM**] [**-c** ["mailer"]] [**-s** newsgroups]

readnews -e [newsgroups]

DESCRIPTION

readnews without argument prints unread articles. There are several user interfaces available:

Flag	Interface
default	A msgs(1) like interface. The header of each article is printed, and the user is asked if the rest of the article should be printed.
-M	An interface to Mail(1). A menu of articles is printed and the user can selectively read them in any order. (This option requires a Mail implementing the -T option.)
-c	A /bin/mail(1) like interface. The text of the article is printed, and at the end, the user is prompted for a command.
-c ``mailer''	The user can use his favorite mailer to read news. All selected articles written to a temporary file. Then the mailer is invoked. Any '%' is substituted with the name of the temporary file. Thus, "mail -f %" will invoke mail on a temporary file consisting of all selected messages.
-p	All selected articles are sent to the standard output. No questions asked.
-l	Only the titles of the selected articles are output.

The -r flag causes the articles to be printed in reverse order.

The following flags determine the selection of articles.

-n newsgroups
Select all articles that belong to newsgroups.

-s newsgroups
Same as -n except that the mandatory subscription group (usually, **general**) is always selected and the .newsrc file is updated (see below).

-t *titles* Select all articles whose titles contain one of the strings specified by *titles*.

-a [*date*] Select all articles (ignore the .newsrc file) that were posted past the given *date* (in getdate(3) format). If no date is given, the dawn of time is assumed.

readnews maintains a .newsrc file in the user's home directory that specifies all news articles already read. It is updated at the end of each reading session in which the -a, -n, -l, or -t options weren't specified.

If the user wishes, he may place an options line in his .newsrc file. This line starts with the word **options** followed by the list of standard options just as they would be typed on the command line. Such a list may include: the **-s** flag along with a newsgroup list; a favorite interface; and/or the **-r** flag.

The **-e** option causes all articles belonging to the specified newsgroups that have expired to be canceled. Newsgroups defaults to the local default, usually **all**.

EXAMPLES

readnews Read all unread articles using the msgs(1) interface. The .newsrc file is updated at the end of the session.

readnews -c ``ed %'' -l
Invoke the ed(1) text editor on a file containing the titles of all unread articles. The .newsrc file is **not** updated at the end of the session.

readnews -s all !fa.all -M -r
Read all unread articles except articles whose newsgroups begin with "fa." via Mail(1) in reverse order. The .newsrc file is updated at the end of the session.

readnews -p >> /usr/spool/mail/*myname*
In effect, mail all recent news to yourself, so that they can be read with your regular electronic mail. This use is discouraged, because it uses extra disk space for the extra copies, and because it does not lock your mailbox.

readnews -p -n all -a last thursday
Print every single article since last Thursday. The .newsrc file is **not** updated at the end of the session.

FILES

/usr/spool/news/*newsgroup*/*number*	News articles
/usr/lib/news/active	Active newsgroups
/usr/lib/news/sys	System subscriptions

/usr/lib/news/help	Help file for msgs(1) interface
~/.newsrc	Options and list of previously read articles

SEE ALSO

newscheck(1), inews(1), sendnews(1), recnews(1), uurec(1), msgs(1), Mail(1), mail(1), getdate(3)

AUTHORS

Matt Glickman (ucbvax!x!glickman)
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California

Mark Horton (cbosg!mark)

Stephen Daniel (duke!swd)
Tom R. Truscott (duke!trt)
Department of Computer Science
Duke University
Durham, North Carolina

NAME

recnews - receive unprocessed articles via mail

SYNOPSIS

recnews [newsgroup [sender]]

DESCRIPTION

recnews reads a letter from the standard input, determines the article title, sender, and newsgroup, and gives the body to inews with the right arguments for insertion.

If newsgroup is omitted, the to line of the letter will be used. If sender is omitted, the sender will be determined from the from line of the letter. The title is determined from the subject line.

SEE ALSO

inews(1), unrec(1), sendnews(1), readnews(1), newscheck(1)

NAME

reboot − replace current UNIX with new program or system

SYNOPSIS

/etc/reboot device file [**nosync**]

DESCRIPTION

Reboot is the command level interface to the *reboot* system call. The device specified should correspond to the octal device code expected by the DEC YC ROM. This number is normally placed in the console switches prior to starting the ROM. The file is the name of the program that is to be booted, replacing the current UNIX. If the **nosync** option is not used this command will link **/dev/syscon** to the terminal from which the command is issued, issue a *sync* system call, halt the currently running UNIX and reboot the specified program. If the **nosync** option is specified *reboot* will verify that the terminal from which the command is issued is linked to **/dev/syscon** before halting the current UNIX and rebooting the specified program.

SEE ALSO

reboot(2)

NAME

reform − reformat text file

SYNOPSIS

reform [tabspec1 [tabspec2]] [**+b***n*] [**+e***n*] [**+f**] [**+i***n*] [**+m***n*] [**+p***n*] [**+s**] [**+t***n*] .

DESCRIPTION

Reform reads each line of the standard input file, reformats it, and then writes it to the standard output. Various combinations of reformatting operations can be selected, of which the most common involve rearrangement of tab characters.

Reform first scans any arguments, which may be given in any order. It then processes its input file, performing the following actions upon each line, in the order given.

- A line is read from the standard input.
- If **+s** is given, the first 10 characters of the line are stripped off. Characters 1−4 (SCCS Release) and 6−9 (SCCS Level) are saved for later addition to the end of the line.
- The line is expanded into a tabless form, by replacing tabs with blanks according the input tab specification *tabspec1*.
- If **+p***n* is given, *n* blanks are prepended to the line.
- If **+t***n* is given, the line is truncated to a length of *n* characters.
- All trailing blanks are now removed.
- If **+e***n* is included, the line is extended out with blanks to the length of *n* characters.
- If **+s** is given, the previously-saved SCCS Release and Level are added to the end of the line.
- If **+b***n* is given, the *n* characters at the beginning of the line are converted to blanks, if and only if all of them are either digits or blanks.
- If **+m***n* is included, the line is moved left, i.e., *n* characters are removed from the beginning of the line.
- The line is now contracted by replacing some blanks with tab characters according to the list of tabs indicated by the output tab specification *tabspec2*, and is written to the standard output file. Option **+i** controls the method of contraction (see below).

The various arguments accepted by *reform* are as follows:

tabspec1 describes the tab stops assumed for the input file. This tab specification may take on any of the forms described below. In addition, the operand −− indicates that the tab specification is to be found in the first line read from the standard input. If no legal tab specification is found there, −8 is assumed. If *tabspec1* is omitted entirely, −− is assumed.

tabspec2 describes the tabs assumed for the output file. It is interpreted the same as *tabspec1*, except that omission of *tabspec2* causes the value of *tabspec1* to be used for *tabspec2*.

The remaining arguments are all optional and may be used in any combination, although only a few combinations make much sense. Specifying an argument causes an action to be performed, as opposed to a usual default of not performing the action. Some options include numeric values, which also have default values. Option actions are applied to each line in the order described above. Any line length mentioned applies to the length of a line just previous to the execution of the option described, and the terminating newline is never counted in the line length.

+b*n* causes the first *n* characters of a line to be converted to blanks, if and only if those characters include only blanks and digits. If *n* is omitted, the default value is 6, which is useful in deleting sequence numbers from COBOL programs.

+e*n* causes each line shorter than *n* characters to be extended out with blanks to that length. Omitting *n* implies a default value of 72. This option is useful for those rare cases in which some sequence numbers need to be added to an existing unnumbered file, e.g., the use of $ in editor regular expressions is more convenient if all lines have equal length. I.e., the user intends to issue editor commands like:

s/$/00001000/

+f causes a format line to be written to the standard output, preceding any other lines written. The format line is taken from *tabspec2*, i.e., the line normally appears as follows:

<:t− *tabspec2* **d:>**

+i*n* controls the technique used to compress interior blanks into tabs. Unless this option is specified, any sequence of 1 or more blanks may be converted to a single tab character if it occurs in an appropriate place. This causes no problems for blanks occurring before the first nonblank character in a line, and it is always possible to convert the result back to an equivalent tabless form. However, occasionally an interior blank (one occurring after the first nonblank) is converted to a tab when this is not intended. For instance, this might occur in any program written in a language utilizing blanks as delimiters. Any single blank might be converted to a tab if it occurred just before a tab stop. Insertion or deletion of characters preceding such a tab may cause it to be interpreted in an unexpected way at a later time. If the **+i** option is used, no string of blanks may be converted to a tab unless there are *n* or more contiguous blanks. The default value is 2. Note that leading blanks are always converted to tabs when possible.

+m*n* causes each line to be moved left *n* characters, with a default value of 6. This can be useful for crunching COBOL programs.

+p*n* causes *n* blanks to be prepended (default of 6 if omitted). This option is effectively the inverse of **+m***n*, and is often useful for adjusting the position of *nroff(1)* output for terminals lacking both forms tractor positioning and settable left margin.

+s is used with the **−m** option of *get*(1S). The **−m** option causes *get* to prepend SCCS Release and Level numbers to each generated line. The **+s** option causes these numbers to be removed from the front of each line, saved, and then later added to the end of the line. Because **+e72** is implied by this option, the effect is to produce 80-character card images with SCCS Release and Level in columns 73−80.

+t*n* causes any line longer than *n* characters to be truncated to that length. If *n* is omitted, the length defaults to 72. Sequence numbers can thus be removed, also permitting additional preceding blanks to be deleted.

Three types of tab specification are accepted for *tabspec:* "canned", repetitive, and arbitrary.

−*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).

−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**.

−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.

n1,n2,...
the arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 20 numbers are allowed. The maximum tab value accepted is 158. 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.

DIAGNOSTICS

All diagnostics are fatal, and the offending line is displayed following the message.
"line too long" if any line exceeds 512 characters (in tabless form).
"not SCCS -m" if first 10 characters of line are not in proper format when **+s** flag used.

EXIT CODES

0 − normal
1 − any error

SEE ALSO

get(1S), nroff(1)

BUGS

Reform is aware of the meanings of Backspace and Escape sequences, so that it can be used as a postprocessor for *nroff*. However, be warned that the **+e**, **+m**, **+t** options only count characters, not positions. Anyone using these options on output containing backspaces or halfline motions will probably obtain unexpected and useless results.

AUTHOR

J. R. Mashey

NAME

release − restore printing of queued line printer jobs

SYNOPSIS

release type item [item]

DESCRIPTION

Release restores a set of line printer requests which were suspended by a *hold* or *restrain* command to a printable state. *Type* (either **user**, **printer**, or **job**) indicates the type of *items* which follow. At least one *item* is required.

In the case of *type***user,** the *items* are login ids of users with queued jobs. For each specified user, *release* restores printing of all the user's line printer jobs, regardless of the queue in which the request resides. Printing begins on a page boundary, two pages before the point of interruption. If the *type* is **printer**, then each *item* is a printer name, and *release* places the named printers back in service of their respective queues. For *type***job,** each of the specified jobs is restored to a printable state.

SEE ALSO

abort(1), init(1), hold(1), lpr(1), restrain(1), start(1)

NAME

restrain — suspend printing of queued line printer jobs

SYNOPSIS

restrain type item [item]*

DESCRIPTION

Restrain suspends printing of a set of line printer requests, but does not interrupt any currently printing job. *Type* (either **user**, **printer**, or **job**) indicates the type of *items* which follow. At least one *item* is required.

In the case of *type***user,** the *items* are login ids of users with queued jobs. For each specified user, *restrain* suspends the printing of all the user's line printer jobs, regardless of the queue in which the request resides. If the *type* is **printer**, then each *item* is a printer name, and *restrain* allows the currently printing job to complete on each of the named printers, but disallows further printing by that printer until it is either *start*ed or *release*d. For *type***job**, each of the specified jobs is inhibited from printing, unless it currently printing.

SEE ALSO

abort(1), init(1), hold(1), lpr(1), release(1), start(1)

NAME

rew – rewind tape

SYNOPSIS

rew [[**m**] digit]

DESCRIPTION

Rew rewinds DECtape or magtape drives. *Digit* is the logical tape number, and should range from 0 to 7. If *digit* is preceded by m, *rew* applies to magtape rather than DECtape. A missing digit indicates drive 0.

FILES

/dev/tap? , /dev/mt?

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** (force) option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument **−r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **−i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **−r**, whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

unlink(2)

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file .. merely to avoid the antisocial consequences of inadvertently doing something like 'rm −r .*'.

NAME

rmdel − remove a delta from an SCCS file

SYNOPSIS

rmdel −rSID name ...

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 SID 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*(1S)) exists for the named SCCS file, the SID 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 pathname 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 exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES

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

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

SEE ALSO

get(1S), delta(1S), prs(1S), help(1S), sccsfile(5)

Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi

DIAGNOSTICS

Use *help*(1S) for explanations.

NAME

rsh — restricted shell (command interpreter)

SYNOPSIS

rsh [flags] [name [arg1 ...]]

DESCRIPTION

Rsh is a restricted version of the standard command interpreter *sh*(1). It 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:

cd
command names containing /
> and >>

When invoked with the name **−rsh**, *rsh* reads the user's **.profile** (from **$HOME/.profile**). It acts as the standard *sh* while doing this, except that an interrupt causes an immediate exit, instead of causing a return to command level. 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 restricting him to 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, then leaving the user in an appropriate directory (probably *not* the login directory). The **.profile** should make **$PATH** *readonly* to keep the user from changing it.

Rsh is actually just a link to *sh* and any *flags* arguments are the same as for *sh*(1).

The system administrator often sets up a directory of commands that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

SEE ALSO

sh(1), profile(5).

NAME

rstlfs − restore logical file system from tape

SYNOPSIS

rstlfs lfs_name tape_unit#

DESCRIPTION

Rstlfs restores a logical file system (LFS) from a tape written by *dmplfs*(1). When used in conjunction with *dmplfs*, *rstlfs* restores a previously-save LFS or completes the compression of a fragmented LFS. After reloading by *rstlfs*, the logical files are stored in ascending order immediately following the overhead area, and the free space is consolidated into a single large area following the last logical file, with the file definition entries, freelist, and bitmap modified accordingly.

Lfs_name is the filename of the LFS within directory **/dev** and *tape_unit#* is the number of the tape drive on which the dump tape is mounted. For convenience, the user may specify the tape unit as 0-3; the program will modify the unit number as necessary to get the correct density. Both parameters are required and if the command is entered without parameters, the program will print the expected syntax.

Rstlfs assumes the LFS to be restored is the same one that was dumped on the tape and checks the tape label to see that the names are the same; the user is asked whether to continue if there is a mismatch. It also checks to be sure that the reels are mounted in the correct order, and prompts the user when a new reel is to be mounted.

FILES

/dev/*lfs_name*	LFS to be read from tape
/dev/mt*tape_unit#*	tape unit to be used
/etc/lmtab	list of mounted logical file systems

SEE ALSO

dmplfs(1), lfcheck(1), mklfs(1)

DIAGNOSTICS

Rstlfs prints self-explanatory error messages on exit whenever a problem is detected.

WARNINGS

If the LFS did not check (using *lfcheck*(1)) prior to running *dmplfs*, any overlapped files will have been "unfolded" and there should be no duplicated blocks after reloading with *rstlfs*.

Do not attempt to restore a mounted logical filesystem; the LFS should be unmounted and flushed to disk before *rstlfs* is invoked.

The LFS should be re-made using *mklfs*(1) before restoring. As additional insurance, it is wise to make a dd tape of the LFS block device before doing the *mklfs* so the LFS can be restored to its prior state if necessary (i.e., if *rstlfs* has trouble reading the *dmplfs* tape).

Rstlfs assumes that the 1600 bpi tape units have file names **/dev/mt8** − **/dev/mt11** (rewind) and **/dev/mt12** − **/dev/mt15** (no rewind).

BUGS

As the program cannot tell when the end of the data file has been reached, the user must enter a **q** instead of just a <CR> to let *rstlfs* know when the last reel has been read.

NAME

sa − shell accounting

SYNOPSIS

sa [**−abcijlmnorstuv**] [file]

DESCRIPTION

When a user logs in, if the Shell is able to open the file **/usr/adm/sh_acct**, then as each command completes, the Shell writes at the end of this file the name of the command, the user, system time and real time consumed, and the user ID. *Sa* reports on, cleans up, and generally maintains this and other accounting files. To turn accounting on and off, the accounting file must be created or destroyed externally. If the user is the super-user, accounting is placed into **/usr/adm/su_acct** instead. As with **sh_acct**, it must be created or destroyed externally. Similarly, if the file **/usr/adm/acct** is created and system accounting is activated (using *accton*(2)), an accounting record is written when each process terminates.

Sa is able to condense the information in **/usr/adm/su_acct** or **/usr/adm/acct** into a summary file **/usr/adm/savacct** which contains a count of the number of times each command was called and the time resources consumed. The summary file **/usr/adm/usracct** is used to record for each user the total number of commands executed, and the total cpu time used. This condensation is desirable because on a large system accounting files can grow by 100 blocks per day. The summary file is read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; **acct** is the default. When a Shell accounting file (**sh_acct** or **su_acct**) is processed, commands that were executed from a command file have an asterisk appended to their name. If the system accounting file, **acct**, is processed, an asterisk is appended to those commands that did not do an *exec*(2) (e.g. daemons). There are many options:

- **a** Do not place all command names containing unprintable characters and those used only once under the name "*****other**".
- **b** Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.
- **c** Besides total user time, system time, and real time for each command, print percentage of total time over all commands.
- **i** Do not read the summary files **/usr/adm/savacct** and **/usr/adm/usracct**, so only information in the accounting files is condensed and reported.
- **j** Instead of total minutes time for each category, give seconds per call.
- **l** Separate system and user time; normally they are combined.
- **m** Superseding all other flags except **u**, print for each user the login name, number of commands executed, and the total number of minutes of cpu time used.
- **n** Sort by number of calls.
- **o** Superseding the **l** flag, report for each command the ratio of user cpu time to user + system cpu time.
- **r** Reverse order of sort.
- **s** Merge accounting file into summary files **/usr/adm/savacct** and **/usr/adm/usracct** when done. This option also causes the **a** option to be used.
- **t** For each command report ratio of real time to the sum of user and system times.
- **u** Superseding all other options, print the raw data.
- **v** If the next character is a digit *n*, then type the name of each command used *n* times or

fewer. Await a reply from the typewriter; if it begins with "**y**", add the command to the category "****junk****". This is used to strip out garbage.

FILES

/usr/adm/sh_acct
/usr/adm/acct
/usr/adm/savacct
/usr/adm/usracct
/usr/adm/su_acct

SEE ALSO

ac(1), accton(1), accton(2)

BUGS

Probably.

NAME

savdate – save and restore modification date

SYNOPSIS

savdate file1 file2 ... : cmd arg1 arg2 ...

DESCRIPTION

Savdate saves the modification dates of each file argument; invokes the command *cmd* with the arguments given; and then restores the original modification dates. The colon (:) must be supplied as a separate argument to allow *savdate* to determine where the file list ends and the command begins.

SEE ALSO

mdate(2), stat(2)

NAME

sccsclean – remove unwanted files in SCCS directories

SYNOPSIS

sccsclean [–n] [–s] file [files]

DESCRIPTION

Sccsclean is a utility to clean out SCCS directories. It will take as arguments a list of files or directories with optional flags. Each directory argument is expanded into a list of all files in the directory. All non-essential files in the resulting list which can be re-created from other files in the current directory are listed on the standard output and removed. This includes all **.o** files and all files which have a corresponding **s.** SCCS source file. Any file which has a corresponding **p.** SCCS protection file is not removed; it is presumed to be the most recent edited copy.

Optional flags consist of:

-n no removal; causes *sccsclean* to merely list the files it finds which would normally be removed.

-s silent; do not list removed files.

Sccsclean is designed to be used in conjunction with *make*(1) to replace the normal "rm *.o" rule for the *clean:* target, so that either normal or SCCS directories are effectively cleaned.

NAME

sccsdiff — compare two versions of an SCCS file

SYNOPSIS

sccsdiff old-spec new-spec [pr-args] file ...

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. The *old-spec* is any valid *get*(1S) specifier (e.g., **-r1.1**) for the old version to be gotten. Similarly, *new-spec* is any valid *get* specifier (e.g., **-r1.4**) for the new version to be gotten. The *pr-args* are any valid *pr*(1) arguments which begin with a **—**, except for **—h** (the output of *sccsdiff* is piped through *pr*). Any number of SCCS files may be specified, but the *old-spec* and *new-spec* apply to all files.

Sccsdiff is a simple shell procedure; interested persons should "cat /usr/bin/sccsdiff" to discover how it works.

FILES

/tmp/get?????	Temporary for old gotten version
/usr/bin/bdiff	Program that generates differences

SEE ALSO

bdiff(1), get(1S), help(1S), pr(1)
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1S) for explanations.

file: No differences If the two versions are the same.

NAME

sccstring – echo SCCS keywords to standard output

SYNOPSIS

sccstring file

DESCRIPTION

The *sccstring* command will echo SCCS keywords %W% to the standard output surrounded by the appropriate characters to make the entire string a comment. *Sccstring* knows the comment conventions for files whose suffices are:

*.[chly], *.mk, [Mm]akefile, *.sh

The output string is a legitimate comment for each of the types of file.

Sccstring is used by the *addsccs*(1S) command to automatically add the SCCS keyword string to named files.

SEE ALSO

addsccs(1S), gadd(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2.

BUGS

There is no way to specify the type of file other than the suffix.

NAME

sendnews - send news articles via mail

SYNOPSIS

sendnews [-o] [-a] [-b] [-n newsgroups] destination

DESCRIPTION

sendnews reads an article from it's standard input, performs a set of changes to it, and gives it to the mail program to mail it to destination.

An `N' is prepended to each line for decoding by uurec(1).

The **-o** flag handles old format articles.

The **-a** flag is used for sending articles via the **ARPANET**. It maps the article's path from uucphost!xxx to xxx@arpahost.

The **-b** flag is used for sending articles via the **Berknet**. It maps the article's path from uucphost!xxx to berkhost!xxx.

The **-n** flag changes the article's newsgroup to the specified newsgroup.

SEE ALSO

inews(1), uurec(1), recnews(1), readnews(1), newscheck(1)

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] [file] ...

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.

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:

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.

An argument denoted *text* 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. An argument denoted *rfile* or *wfile* 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.

(1) i\
text

Insert. Place *text* on the standard output.

(2) n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) N Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) p Print. Copy the pattern space to the standard output.

(2) P Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) q Quit. Branch to the end of the script. Do not start a new cycle.

(2) r *rfile*

Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) s/*regular expression*/*replacement*/*flags*

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed*(1). *Flags* is zero or more of:

g Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Print the pattern space if a replacement was made.

w *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) t *label*

Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a t. If *label* is empty, branch to the end of the script.

(2) w *wfile*

Write. Append the pattern space to *wfile*.

(2) y/*string1*/*string2*/

Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! *function*

Don't. Apply the *function* (or group, if *function* is {) only to lines *not* selected by the address(es).

(0) : *label*

This command does nothing; it bears a *label* for b and t commands to branch to.

(1) = Place the current line number on the standard output as a line.

(2) { Execute the following commands through a matching } only when the pattern space is selected.

(0) An empty command is ignored.

SEE ALSO

awk(1), ed(1), grep(1).
SED−A Non-interactive Text Editor by L. E. McMahon.

NAME

sema — semaphore operations

SYNOPSYS

sema {**p**|**v**|**test**|**post**|**block**|**setsem**|**lock**|**unlock**|**tlock**|**rdsem**} sema-number {newval}

DESCRIPTION

Sema is used to perform semaphore operation from the Shell. The second argument to the command specifies which semaphore operation is to be performed on the third argument, which specifies the semaphore number. In all cases the value of the semaphore before the operation was performed is reported.

SEE ALSO

Sema(2)

NAME

setpgrp – execute program with new process group

SYNOPSIS

setpgrp [[pgrp] command [argument ...]]

DESCRIPTION

Setpgrp executes the specified program after changing the process' group to process ID of the command itself (to guarantee uniqueness). If the optional *pgrp* argument is given, the process group is set to the group specified. *Setpgrp* with no arguments will report the current process group. This command is normally used to spin commands off from a terminal when it is desired to ignore any future signals generated at the terminal.

SEE ALSO

setpgrp(2)

NAME

sh, rsh − shell, the standard/restricted command programming language

SYNOPSIS

sh [−**ceiknrstuvx**] [args]
rsh [−**ceiknrstuvx**] [args]

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh;* it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

Commands.

A *simple-command* is a sequence of non-blank *words* separated by *blanks* (a *blank* is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by |. The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**
: Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**
: A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation* below).

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
: The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**
: A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a sub-shell.

{*list*;}

list is simply executed.

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments.
A word beginning with **#** causes that word and all the following characters up to a new-line to be ignored.

Command Substitution.
The standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution.
The character **$** is used to introduce substitutable *parameters*. Positional parameters may be assigned values by **set**. Variables may be set by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*.

${*parameter***}**

A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters *, @, #, ?, −, $, and !. The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A *name* must begin with a letter or underscore. If *parameter* is a digit then it is a positional parameter. If *parameter* is * or @, then all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter***:−***word***}**

If *parameter* is set and is non-null then substitute its value; otherwise substitute *word*.

${*parameter***:=***word***}**

If *parameter* is not set or is null then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter***:?***word***}**

If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, then the message "parameter null or not set" is printed.

${*parameter***:+***word***}**

If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:−`pwd`}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
−	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under rsh.

CDPATH The search path for the *cd* command.

MAIL If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file.

TIMEO Time interval for shell timeout, decimal seconds (see *Timeout* below).

PS1 Primary prompt string, by default "**$** ".

PS2 Secondary prompt string, by default "> ".

IFS Internal field separators, normally **space**, **tab**, and **new-line**.

The shell gives default values to **PATH**, **TIMEO**, **PS1**, **PS2**, and **IFS**, while **HOME** and **MAIL** are not set at all by the shell (although **HOME** *is* set by *login*(1)).

Blank Interpretation.
After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation.
Following substitution, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

* Matches any string, including the null string.

? Matches any single character.

[...] Matches any one of the enclosed characters. A pair of characters separated by − matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "!" then any character not enclosed is matched.

Quoting.
The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | < > new-line space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **\new-line** is ignored. All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, `, ", and **$**. **"$*"** is equivalent to **"$1 $2 ..."**, whereas **"$@"** is equivalent to **"$1" "$2" ....**

Prompting.
When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of **PS2**) is issued.

Input/Output.
Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs

before *word* or *digit* is used:

<word	Use file *word* as standard input (file descriptor 0).
>word	Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.
>>word	Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<<[−]word	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\new-line** is ignored, and \ must be used to quote the characters \, **$**, `, and the first character of *word*. If − is appended to <<, then all leading tabs are stripped from *word* and from the document.
<&digit	The standard input is duplicated from file descriptor *digit* (see *dup*(2)). Similarly for the standard output using >.
<&−	The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by **&** then the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment.

The *environment* (see *environ*(7)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd args                    and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

If the **−k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c:**

```
echo a=b c
set −k
echo a=b c
```

Signals.

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Timeout
The shell parameter **TIMEO** defines the decimal number of seconds which the shell will wait for input after issuing a prompt. If no (possibly null) command has been input during this interval, the warning message

Shell timeout: type return within 30 seconds.

will appear on the terminal. If no return (or command) is input, the shell will terminate 30 seconds later.

A value of **0** for **TIMEO** will result in no timeout.

Execution.
Each time a command is executed, the above substitutions are carried out. Except for the *Special Commands* listed below, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / then the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a sub-shell.

Special Commands.
The following commands are executed in the shell process and, except as specified, no input/output redirection is permitted for such commands:

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / then the search path is not used. Otherwise, each directory in the path is searched for *arg*. **cd** may not be executed by rsh.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted then the exit