NAME

hpm – Monitors hardware performance during program execution

SYNOPSIS

hpm [-d] [-g *group*] [-o *file*] [-p] [-r] [-V] *program* [*args*]

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The hpm utility monitors machine performance while your program executes. Your program can be written in any language available under the UNICOS operating system, though it cannot make use of Parallel Virtual Machine (PVM) message-passing software. hpm gives results for only whole programs and writes its output to standard error.

The hpm utility accepts the following options:

-d Displays additional rates as though run in dedicated mode; some rates are based on wall-clock time, not CPU time.

These times give a rough estimate of concurrency when the user's program is autotasked, microtasked, or macrotasked. The displayed rates should not be considered to be exact. The extra wall-clock time required to execute the code to be monitored, combined with the extra wall-clock time needed to notify the hpm utility that the user's command is done, can easily exceed the wall-clock time needed to actually execute the code.

If you need more precise measurements of execution time versus wall-clock time, use the atexpert(1) command or the ja(1) command. It is possible to run your program on a dedicated machine or under the ded(8) command. However, even running under ded or on a dedicated machine still cannot guarantee exact dedicated timings.

-g *group* Number of the hardware monitor group to be used. This option does not apply to the CRAY C90 or the CRAY T90 series because they have only one Hardware Performance Monitor (HPM) group. The *group* argument can be one of the following values:

- 0 Execution summary (default)
- 1 Hold issue conditions
- 2 Memory activity
- 3 Vector events and instruction summary

-o *file* By default, hpm writes its output to standard error. If this option is specified, hpm writes its output to the file name specified as *file*.

-p Displays results for just the program and excludes all performance information for any child processes.

`-r` Generates a raw-format output, suitable for postprocessing by tools such as `perfview`(1) and `awk`(1). See the *Guide to Parallel Vector Applications*, Cray Research publication SG–2182, for a description of this format.

`-V` Displays the current version of `hpm`, as well as a short copyright notice.

program Executable file to be run.

args Arguments to *program*.

NOTES

HPM does not work with Parallel Virtual Machine (PVM) code on Cray PVP systems.

The default counter hardware monitor group is 0, the group most commonly run by typical users (does not apply to the CRAY C90 or CRAY T90 series).

The performance utility Perftrace also uses the hardware performance monitor device. Using the `libperf.a` library with `hpm` may generate unusable results for both.

On Cray PVP (except CRAY C90 and CRAY T90 series) systems, groups 0 and 3 report megaflop rates. By default, these megaflop rates do not reflect concurrent execution of an autotasked, microtasked, or macrotasked program, since they are calculated per CPU second and not per wall-clock second. When running a multitasked program, you can obtain more informative rates that use wall-clock seconds by specifying the `-d` option. However, note the special considerations for using this option, as described under the `-d` option previously.

The meanings of the HPM statistics and their implications are discussed in detail in the *Guide to Parallel Vector Applications*, Cray Research publication SG–2182.

Because of variations in system overhead and other factors, `hpm` statistics may not be precisely repeatable. The statistics gathered and displayed by the `hpm` utility should not be construed as accounting information, nor should they be considered to be exact.

EXAMPLES

Example 1: For Cray PVP systems, the following examples execute a program four times to receive all four hardware monitor groups in the file `prog.hpm`. The last example shows that only one program execution is needed on the CRAY C90 and CRAY T90 series.

Standard shell or Korn shell (except on the CRAY C90 or CRAY T90 series):

```
$ f90 prog.f
$ hpm -g0 ./a.out 2>> prog.hpm
$ hpm -g1 ./a.out 2>> prog.hpm
$ hpm -g2 ./a.out 2>> prog.hpm
$ hpm -g3 ./a.out 2>> prog.hpm
```

C shell (except on the CRAY C90 or CRAY T90 series):

```
% f90 prog.f
% ( hpm -g0 ./a.out ) > & prog.hpm
% ( hpm -g1 ./a.out ) >> & prog.hpm
% ( hpm -g2 ./a.out ) >> & prog.hpm
% ( hpm -g3 ./a.out ) >> & prog.hpm
```

CRAY C90 or CRAY T90 series (all shells):

```
$ f90 prog.f
$ hpm ./a.out
```

Example 2: The following standard shell example (not for the CRAY C90 or CRAY T90 series) generates raw-format data and then processes the output with the perfview(1) command. After these command lines, perfview(1) runs interactively.

```
$ cc prog.c
$ hpm -g0 -r ./a.out 2> raw.data
$ hpm -g3 -r ./a.out 2>> raw.data
$ perfview raw.data
```

Example 3: The following example (not for the CRAY C90 or CRAY T90 series) writes the hpm data to a file for processing by perfview(1):

```
$ cc prog.c
$ hpm -g0 -o data.0 -r ./a.out
$ hpm -g3 -o data.3 -r ./a.out
$ cat data.0 data.3 >perf.data
$ perfview
```

SEE ALSO

atexpert(1), awk(1), csh(1), ja(1), perfview(1), sh(1)

hpm(4) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

performance(7), perftrace(7) (available only online)

ded(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

Guide to Parallel Vector Applications, Cray Research publication SG–2182, for descriptions of all the performance tuning tools

The following manuals are Cray Research Proprietary; dissemination of this documentation to non-CRI personnel requires approval from the appropriate vice president and a nondisclosure agreement. Export of technical information in this category may require a Letter of Assurance.

NAME

iconv – Codeset conversion

SYNOPSIS

iconv -f *fromcode* -t *tocode* [*file...*]

IMPLEMENTATION

All Cray Research systems

STANDARDS

XPG4

DESCRIPTION

The iconv utility converts the encoding of characters in *file* from one codeset to another and writes the results to standard output.

Character encodings in either codeset may include single-byte values (for example, for the ISO 8859-1:1987 standard characters) or multibyte values (for example, for certain characters in the ISO 6937:1983 standard). The results of specifying invalid characters in the input stream (either those that are not valid members of the *fromcode* or those that have no corresponding value in *tocode*) are specified in the system documentation.

The iconv utility accepts the following options and operands:

-f *fromcode* Identifies the codeset of the input file. Valid values for *fromcode* are specified in the system documentation.

-t *tocode* Identifies the codeset to be used for the output file. Valid values for *tocode* are specified in the system documentation.

file A path name of the input file to be translated. If *file* is omitted, the standard input is used.

The following environment variables affect the execution of iconv:

LANG Provides a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL If set to a nonempty string value, overrides the values of all the other internationalization variables.

LC_CTYPE Determines the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multibyte characters in arguments). During translation of the file, this variable is superseded by the use of the *fromcode* option-argument.

LC_MESSAGES Determines the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determines the location of message catalogs for the processing of LC_MESSAGES.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

EXAMPLES

The following example converts the contents of file mail.x400 from the ISO 6937:1983 standard codeset to the ISO 8859-1:1987 standard codeset, and stores the results in the file mail.local:

```
iconv -f IS6937 -t IS8859 mail.x400 > mail.local
```

SEE ALSO

gencat(1)

NAME

`id` – Prints user and group IDs and names

SYNOPSIS

`id` [*user*]
`id -G` [`-n`] [*user*]
`id -g` [`-n`] [`-r`] [*user*]
`id -u` [`-n`] [`-r`] [*user*]
`id -a` [*user*]

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4
CRI extensions (`-a` option)

DESCRIPTION

If no *user* is specified, the `id` utility writes the user and group IDs and the corresponding user and group names of the invoking process to standard output. When the effective and real IDs do not match, both are written.

If *user* is specified and the invoking process has the appropriate privileges, the user and group IDs of the selected user are written. In this case, effective IDs are assumed to be identical to real IDs.

If multiple groups are supported by the underlying system, the supplementary group affiliations are also written.

The `id` utility accepts the following options:

`-a` Prints the current account ID and name.

`-g` Prints only the effective group ID.

`-G` Prints all different group IDs (effective, real, and supplementary) only. If more than one distinct group affiliation exists, the utility prints each affiliation.

`-n` Prints the name of the user or group, rather than the numeric ID.

`-r` Prints the real ID, rather than the effective ID.

`-u` Prints the only the effective user ID.

NOTES

If this utility is installed with a privilege assignment list (PAL), a user with one of the following active categories is allowed to perform the actions shown:

Active Category	Action
`system`, `secadm`	In a privileged administrator shell environment, allowed to write shell-redirected output to any file.
`sysadm`	Shell-redirected output is subject to security label restrictions.

If the `PRIV_SU` configuration option is enabled, the super user can write shell-redirected output to any file.

WARNINGS

The `id -Gn` command produces the same output as `/usr/ucb/groups`. Users should use the `id` utility because the `groups` utility may be removed in a future UNICOS release.

EXIT STATUS

The `id` utility exits with one of the following values:

0 Successful completion.

>0 An error occurred.

EXAMPLES

The following example prints your user and group IDs and names:

```
$ id
uid=1054(mary) gid=101(acct) groups=508(allgrp),717(compilers),24(source)
```

FILES

`/etc/udb`	User validation file that contains user control limits
`/etc/group`	Group files that contain group names and group IDs

SEE ALSO

`groups`(1B)

`getuid`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

`group`(5), `udb`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

ipcrm – Removes a message queue, semaphore set, or shared memory ID

SYNOPSIS

ipcrm [-m *shmid*] [-M *shmkey*] [-q *msqid*] [-Q *msgkey*] [-s *semid*] [-S *semkey*]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The ipcrm command removes one or more message queues, semaphore sets, or shared memory identifiers.

The ipcrm command accepts the following options:

-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 operation.

-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 operation.

-q *msqid* Removes the message queue identifier *msqid* from the system and destroys the message queue 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.

-s *semid* Removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.

-S *semkey* Removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the remove operations are described in msgctl(2), shmctl(2), and semctl(2). Use the ipcs(1) command to find the identifiers and keys.

NOTES

If this utility is installed with a privilege assignment list (PAL), a user with one of the following active categories is allowed to perform the actions shown:

Active Category	Action
system, secadm	Allowed to remove any identifier.
sysadm	Allowed to remove any identifier, subject to security label restrictions on the identifier's path. Shell-redirected I/O is subject to security label restrictions.

If the PRIV_SU configuration option is enabled, the super user is allowed to remove any identifier.

SEE ALSO

ipcs(1)

msgctl(2), msgget(2), msgrcv(2), msgsnd(2), semctl(2), semget(2), semop(2), shmat(2), shmctl(2), shmdt(2), shmget(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

stdipc(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

ipc(5), msg(5), sem(5), shm(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

ipc(7) Online only

NAME

ipcs – Reports interprocess communication (IPC) facilities status

SYNOPSIS

ipcs [-a] [-b] [-c] [-e] [-m] [-o] [-p] [-q] [-s] [-t]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The ipcs command prints information about active interprocess communication (IPC) facilities. Without options, information is printed in short format for message queues, shared memory, and semaphore sets that are currently active in the system.

The information that is displayed is controlled by the options supplied.

-m Prints information about active shared memory segments.

-q Prints information about active message queues.

-s Prints information about active semaphore sets.

If -q, -m, or -s are specified, information about only those indicated is printed. If none of these three are specified, information about all three is printed subject to the following options. For detailed information about an ipcs listing, see the ipcs Listing Information section.

-a Uses all print options. (This is a shorthand notation for -b, -c, -o, -p, and -t.)

-b Prints information on biggest allowable size: 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.

-c Prints creator's login name and group name.

-e Provides the access control list (ACL) flag, security level, and compartment flag as the fields immediately following the mode field. When at least one compartment is set, the facility's compartment flag is displayed as a plus sign (+) adjacent to the facility's security level. When a facility has an associated ACL, its ACL flag appears as a letter a adjacent to the mode field. A facility that has a wildcard security level has an asterisk (*) displayed for its security level.

-o Prints information on outstanding usage: number of messages on the queue and total number of bytes in those messages and number of processes attached to shared memory segments.

-p Prints process number information: process ID of last process to send a message, process ID of last process to receive a message on message queues, process ID of creating process, and process ID of last process to attach or detach on shared memory segments.

`-t` Prints time information: time of the last control operation that changed the access permissions for all facilities, time of last `msgsnd`(2) and last `msgrcv`(2) on message queues, time of last `shmat`(2) and last `shmdt`(2) on shared memory, time of last `semop`(2) on semaphores.

`ipcs` Listing Information

This section lists the column headings in an `ipcs` listing and describes the information produced by the `ipcs` command. The default headings and informtion produced by this command are as follows except for those described with options in parentheses (for example, `CREATOR`). In these exceptions, the options named cause the corresponding heading to appear.

Heading	Description
`T`	Type of the facility: `q` Message queue `m` Shared memory segment `s` Semaphore
`ID`	The identifier for the facility entry.
`KEY`	The key used as an argument to `msgget`(2), `semget`(2), or `shmget`(2) 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`	The facility access modes and flags: The mode consists of 12 characters that are interpreted as follows. The first three characters are one of the following: `P` The persistent facility is enabled for the message queue, semaphore set, or shared memory segment. `R` A process is waiting on a `msgrcv`(2) operation. `S` A process is waiting on a `msgsnd`(2) operation. `-` The corresponding special flag is not set. The next 9 characters are interpreted as three sets of 3 bits each. The first set refers to the owner's permissions; the next set refers to permissions of others in the user group of the facility entry; and the last set refers 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` Read permission is granted. `w` Write permission is granted. `a` Alter permission is granted. `-` The indicated permission is not granted.

OWNER The login name of the owner of the facility entry.

GROUP 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.

NOTES

Things can change while `ipcs` is running; the information it gives is guaranteed to be accurate only when it was retrieved.

Only an appropriately authorized user can see output for IPC facilities whose active security label is greater than that of the user.

If this command is installed with a privilege assignment list (PAL), a user who is assigned the following privilege text upon execution of this command is allowed to perform the action shown:

Privilege Text	Action
`showall`	Allowed to see output for all IPC facilities.

If the PRIV_SU configuration option is enabled, the super user is allowed to see output for all IPC facilities.

FILES

/dev/Rmem	Kernel data structures
/etc/group	Group names
/etc/passwd	User names
/etc/udb	User database (UDB) information
/etc/udb.public	User database (UDB) public information

SEE ALSO

ipcrm(1)

msgctl(2), msgget(2), msgrcv(2), msgsnd(2), semctl(2), semget(2), semop(2), shmat(2), shmctl(2), shmdt(2), shmget(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

stdipc(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

ipc(5), msg(5), sem(5), shm(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

ipc(7) Online only

NAME

`ispell` – Corrects spelling for a file

SYNOPSIS

`ispell` [*file...*]
`ispell` [`-l` | `-D` | `-E`]
`spell` [`+local_file`] [*file...*]

IMPLEMENTATION

All Cray Research systems

STANDARDS

FSF

DESCRIPTION

The `ispell` program helps you to correct typos in a file, and to find the correct spelling of words. When presented with a word that is not in the dictionary, `ispell` offers possibilities.

The best way to use `ispell` is with GNU Emacs. For documentation about this mode, see the info topic "`ispell`."

`ispell` can also be used by itself. In this case, the most common usage is "`ispell` *filename*." If `ispell` finds a word that is not in the dictionary, the word is printed at the top of the screen. `ispell` then checks the dictionary for near misses (words that differ only by a single letter, a missing or extra letter, or a pair of transposed letters). Any that are found are printed on the following lines, and two lines of context containing the word are printed at the bottom of the screen. If your terminal can display reverse video, the word is highlighted.

If you think the word is correct as is, you can press the `<space>` key to accept it this one time, the `<a>` key to accept it for the rest of this file, or the `<i>` key to accept it and put it in your private dictionary. If one of the near misses is the word you want, type the corresponding number. You can press the `<r>` key and you will be prompted for a replacement word. The string you type will be broken into words, and each one will also be checked. You can also press the `<?>` key for help.

`ispell` accepts the following options:

`-l` Produces a list of misspelled words from the standard output. This mode is compatible with the traditional spell program, except that the output is not sorted.

`-D` Prints words with flags.

`-E` Prints expanded words as follows:

```
% ispell
word: independant
how about: independent
word: ^D
```

`-u` `ispell` tries to be compatible with the traditional spell program.

file Name of file to be corrected.

If `ispell` is started with no arguments, it enters a loop reading words from the standard input and printing messages about them on the standard output. You can use this mode to find the spelling of a problem word.

There are several other options provided so that other programs can use `ispell`. See the documentation in the `ispell` source directory for details.

If `ispell` is executed by using the name `spell`, it tries to be compatible with the traditional spell program. You can also get this behavior by specifying the `-u` option. In this case, the list of files (or standard input) is checked, and an alphabetized list of misspellings is produced on the standard output.

FILES

`/usr/lib/ispell/ispell.dict`	System dictionary
`$HOME/ispell.words`	Private dictionary

SEE ALSO

`emacs`(1)

`/usr/lib/emacs/info/ispell.texinfo`
/

NAME

ja – Job accounting information

SYNOPSIS

ja [[-f] [-o] [-s [-e]] [[-a *acid*] [-d] [-D] [-g *d*] [-j *jid*] [-l [-C] [-h]] [-n *names*] [-p *marks*] [-r] [-u *uid*]]] [-t] [*file*]

ja [-m] [*file*]

ja [[-c[-h]] [-f] [-s [-e]] [[-a *acid*] [-d] [-D] [-g *gid*] [-j *jid*] [-l[-C-h]] [-n *names*] [-p *marks*] [-r] [-u *uid*]]] [-t] [*file*]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The ja command provides job- or session-related accounting information. This information is taken from the job accounting file to which the kernel writes, provided that job accounting is enabled. The job accounting file can be either the *file* you provide or the default, described in the following. ja provides information only about terminated processes. The login shell and the current ja command being executed are active processes and are not reported by ja. See ps(1) for information about active processes.

To enable job accounting, use the ja command. You may specify only the mark option (-m) and the optional *file* name when enabling. If the job accounting file does not exist, ja creates it. If the file does exist, accounting information is appended to the existing file. If job accounting is already enabled and the optional file name specified is a file other than the currently active job accounting file, the newly specified file becomes the job accounting file.

If you do not specify the optional file name, a default name of the following form is used:

$TMPDIR/.jacct*job ID*

The TMPDIR environment variable is not exported in at(1) or crontab(1) jobs. You must specify the job accounting file name in the at(1) or crontab(1) commands; otherwise, ja will abort.

On normal termination of job accounting (-t specified), ja removes the job accounting file and disables job accounting. If you specify the optional file name when enabling, specify the same name when terminating.

The ja command lets you mark the positions of various commands (processes) by writing the position of the next accounting record to be processed to standard output. You can use these marks when generating reports to restrict the information reported.

There are three groups of options you can use with the `ja` command:

Report selection options
([[`-c`], [`-f`], [`-o`], and [`-s`])
Mark and disable options
([`-m`] or [`-t`])
Report modifier options
([`-d`], [`-e`], [`-h`], [`-l`], [`-r`], [[`-a` *acid*], [`-g` *gid*], [`-j` *jid*], [`-n` *names*], [`-p` *marks*], [`-u` *uid*]]], [`-C`], and [`-D`])

Report Selection Options

The `ja` command can produce four kinds of reports by using the `-c`, `-f`, `-o`, and `-s` options; these are first summarized and then described in detail.

In summary, the four report selection options are as follows:

`-c` Produces the command report (`-c` and `-o` are mutually exclusive).

`-f` Produces the command flow report.

`-o` Produces the other (alternative) command report (`-o` and `-c` are mutually exclusive).

`-s` Produces the summary report.

Described in detail, the four report selection options are as follows:

`-c` Produces a command report. The following fields are reported when you specify the `-c` option with the `-l` option or with the `-l` and `-h` options. These fields provide statistics about individual processes.

`Command Name`	First 8 characters of the name of the command that was executed.
`Started At`	Start time of the process.
`Elapsed Seconds`	Elapsed time of the process.
`User CPU Seconds`	Amount of CPU time the process consumed while it was executing in user mode.
`Sys CPU Seconds`	Amount of CPU time the process consumed while it was executing in system mode.
`I/O Wait Sec Lck`	Amount of time the process waits for I/O while it is locked in memory. I/O wait time is the time a process is blocked until it is rescheduled. The process is blocked while waiting for things such as raw I/O to complete.
`I/O Wait sec Unlck`	Amount of time the process is blocked until it is rescheduled while it is not locked in memory. Time spent for system buffers and buffered I/O blocks are included.

`CPU Mem Avg Mwds` Average amount of memory that this process used. This value is calculated by dividing the memory integral by the total CPU time (*system + user CPU time*). For more information about memory integrals, see *UNICOS Resource Administration*, Cray Research publication SG–2302.

`I/O WMem Avg Mwds` Average amount of memory that the process used while it was locked in memory and waiting for I/O. The value is calculated by dividing the I/O wait memory integral by the I/O wait time while locked in memory.

`Kwords Xferred` Number of characters read or written by the `read`, `write`, `reada`, `writea`, and `listio` system calls (see `read`(2), `write`(2), `reada`(2), `writea`(2), and `listio`(2)).

`Log I/O Request` Number of logical I/O requests that the process performed. A logical I/O request is performed each time a process calls a `read`, `write`, `reada`, or `writea` system call. When the `listio` system call (see `listio`(2)) is called, the number of logical I/O requests is equal to the number of strides multiplied by the number of requests processed.

`Phy I/O Request` Number of times data actually was read from or written to a device. This count does not include requests found in the buffer cache and requests retrieved along with another I/O request.

`Memory HiWater` Maximum amount of memory the process used at any one time. The value is reported in units of 512 words.

`Ex St` Lower 8 bits from the exit status of the process. See `wait`(2) for more information.

`Ni` The last nice value of the process; reported when the total CPU time (*user + system CPU time*) is less than 1 second. If the total CPU time is greater than or equal to 1 second, the minimum nice value at 1 second and onward is listed. The minimum nice value corresponds to the highest priority the process was niced.

`Fl` Accounting flag. The following values are available:

`F` The process forked but did not execute.

`S` The process used super-user privileges.

The accounting flags are defined in `/usr/include/sys/acct.h`.

`SBUs` System billing unit (SBU) for the process. The system administrator configures SBU calculations. For more information, see *UNICOS Resource Administration*, Cray Research publication SG–2302.

When a process is multitasked, the `-c` option produces reports that contain the following fields. Additional lines of multitasking information follow the per-process statistics.

`Command Name` Number of processors that were connected to the process. For example, a value of `# 1 CPU` indicates that the line contains information about the process when only one CPU was connected to it. Likewise, `# 2 CPU` denotes information when two CPUs were connected.

`User CPU Seconds` User CPU time (in seconds) when the process was connected to the number of CPUs specified in the `Command Name` field. The user CPU time reported in this field includes wait semaphore time.

Example:

```
Command     Started     Elapsed     User CPU
 Name          At       Seconds      Seconds
-------     -------     -------     --------

 a.out     16:19:18     1.1251       5.6487
# 1 CPU                             0.0122
# 2 CPU                             0.0194
# 3 CPU                             0.0125
# 4 CPU                             0.0028
# 5 CPU                             0.0120
# 6 CPU                             0.0108
# 7 CPU                             0.8371
# 8 CPU                             4.7813
```

In this example, `a.out` is a multitasked program. The sum of the multitasked user CPU time is `5.6881`, which is larger than `5.6487`, the user CPU time reported for `a.out`. The larger number reflects the fact that the multitasked user CPU time includes wait semaphore time.

To calculate the wait semaphore time, subtract the per-process user CPU time from the sum of the multitasked user CPU times. In the previous example, the wait semaphore time is 5.6881 – 5.6487 = 0.0394 seconds.

The `-c` and `-d` options produce the following additional fields that contain information about device-specific I/O for the process in the preceding line.

`Command Name` Logical device accounting name. The name may span many fields. An example value is `# Block device dd29`.

`Kwords Xferred` Number of characters read or written by the `read`(2), `write`(2), `reada`(2), `writea`(2), and `listio`(2) system calls to the device specified in the `Command Name` field.

`Log I/O Request` Number of the `read`(2), `write`(2), `reada`(2), and `writea`(2) system calls made to the device. When `listio`(2) is called, the number of logical I/O requests is equal to the number of strides multiplied by the number of requests processed on the device.

`-f` Produces a command flow report. This report provides information on the parent/child relationships of processes and, if you specify the `-l` option, CPU user and system time (in seconds).

`-o` Produces an alternative (other) command report. The `-o` option report contains the following fields, which show statistics about individual processes. Several fields show significant values only if performance accounting has been enabled; otherwise, the string `NA` is printed.

`Command Name`	First 8 characters of the name of the command that was executed.
`Started At`	Start time of the process.
`Elapsed Seconds`	Elapsed time of the process.
`Proc ID`	Process ID of the current process.
`Parent ProcID`	Process ID of the parent process.
`Sys Call Seconds`	System call time (in seconds).
`I/O Wait Secs Term`	I/O terminal wait time; I/O wait time is the period of time starting when a process is blocked and ending when it is rescheduled. This field contains a significant value only if performance accounting is enabled (see `devacct`(8)).
`Wait Swap Seconds`	Time (in seconds) that the process waited while swapped out of memory. This field contains a significant value only if performance accounting is enabled (see `devacct`(8)).
`Number of Swaps`	Number of swaps for the current process.
`Phy Blks Mvd: Buf`	Number of physical blocks transferred by the process from or to a block device by using the system buffer I/O interface. This field contains a significant value only if performance accounting is enabled (see `devacct`(8)).
`Phy Blks Mvd: Raw`	Number of physical blocks transferred by the process to and from a block device by using the raw I/O interface. This field contains a significant value only if performance accounting is enabled (see `devacct`(8)).
`Memory Hiwater`	Maximum amount of memory the process used at any one time. The value is reported in units of 512 words.
`Start fract`	Clocks since last second mark was displayed. This field contains a significant value only if performance accounting is enabled (see `devacct`(8)).

`-s` Produces a summary report. The `-s` option report contains the following fields, which provide accumulated usage statistics for the reporting period.

`Job Accounting File Name`
: Name of the file to which the kernel writes the `ja` accounting records.

`Operating System`
: Operating system name, node name, release, version, and hardware type.

`User Name (ID)`
: Name and user ID of the real user.

`Group Name (ID)`
: Name and group ID of the real group.

`Account Name (ID)`
: Account name and account number that this process uses. Multiple account ID usage is listed, but not individual accounts.

`Job ID`
: Job ID associated with these processes.

`Report Starts`
: Starting time of the process that began first during the reporting period.

`Report Ends`
: Ending time of the process that was the last to complete during the reporting period.

`Elapsed Time`
: Duration of the reporting period in seconds (the difference between the report ending and starting times).

`User CPU Time`
: Total CPU time (in seconds) used during the reporting session while the processes were in user mode. (This field is expanded to report multitasking data.)

`System CPU Time`
: Total CPU time (in seconds) used during the reporting session while the processes were in system mode.

`I/O Wait Time (Locked)`
: Cumulative time (in seconds) the system spent waiting for I/O while the processes were locked in memory.

`I/O Wait Time (Unlocked)`
: Cumulative time (in seconds) the system spent waiting for I/O while the processes were not locked in memory.

`CPU Time Memory Integral (Mword-second)`
: Sum of the memory integrals for all processes. For more information on memory integrals, see *UNICOS Resource Administration*, Cray Research publication SG–2302.

`SDS Time Memory Integral`
: (Not on CRAY EL series systems) Measure of SDS use with respect to how long the SDS space was used.

`I/O Wait Time Memory Integral (Mword-second)`
: Measure of how much memory was used when the processes waited for I/O while locked in memory.

`Data Transferred` Total number of characters read or written by the `read`(2), `write`(2), `reada`(2), `writea`(2), and `listio`(2) system calls by all processes in the reporting period.

`Maximum memory used (Mword)`
Maximum amount of memory used by any process at one time.

`Logical I/O Requests`
Total number of `read`(2), `write`(2), `reada`(2), and `writea`(2) system calls executed by all processes in the reporting period. The sum of the number of strides multipled by the number of requests processed for each `listio`(2) call is added to the logical I/O request total.

`Physical I/O Requests`
Total number of times data was read to or written from a physical device by all processes in the reporting period.

`Number of Commands` Total number of commands that completed during the reporting period.

`Billing Units` Sum of the system billing units (SBUs) of all processes.

If a process uses a massively parallel processor (where there is a Cray MPP system), the report contains the following additional information:

`MPP Time` Total number of CPU seconds that the Cray MPP system was used.

`MPP Barrier Bits` Total number of barrier bits used and the largest number used at one time.

`MPP Processor Elements`
Total number of processing elements (PEs) used and the largest number used at one time.

If a process is multitasked, the `User CPU Time` section of the report using the `-s` option expands to include the following information:

`User CPU Time` Total amount of user CPU time (seconds) used during the reporting session while the processes were in user mode.

If the system administrator has defined weighting factors for multiple CPU use, a weighted user CPU time is also reported. This value is in brackets. It is calculated by taking the sum of the weight multiplied by the user CPU time for all concurrent CPUs used. In this instance, the user CPU time includes the wait semaphore time.

In Example 3 of the EXAMPLES section, the total user CPU time (excluding wait semaphore time) is 3.5826 seconds. The weighted user CPU time is 3.4636 seconds. Therefore, an incentive exists to use multitasking with this program.

`Multitasking Breakdown`
: The first part of the breakdown shows the user CPU usage by the number of concurrent CPUs used. The `Weight` and `Weighted seconds` columns appear only if the system administrator has defined CPU weighting factors.

`Concurrent CPUs`
: Number of processors simultaneously connected to the process(es). A connected processor may be executing real work, or it may be waiting on a blocked condition, such as a semaphore.

`Weight`
: The weighting factor for having *n* CPUs connected to the process(es), where *n* is given by the previous field. To determine the factor for the *n*th CPU, subtract the current weighting factor from the one in the previous line, if there is one. In Example 3 of the EXAMPLES section, the second CPU is 90% as expensive as the first CPU.

`Connect Seconds`
: The number of seconds that N CPUs were connected to the processes.

`CPU Seconds`
: The number of user CPU seconds used by *n* concurrent processors. It includes wait semaphore time and is the product of the number of concurrent CPUs multiplied by the connect seconds.

`Weighed Seconds`
: Weighted CPU seconds; the product of the weight and the CPU seconds.

The final portion of the breakdown shows multitasking summary statistics.

`Concurrent CPUs (Avg.)`
: Average number of concurrent CPUs that were connected to the process(es). It is calculated by dividing the total CPU seconds by the total connect seconds.

`Weight (Avg.)`
: Average weighting factor. It is calculated by dividing the total weighted seconds by the total connect seconds.

`Connect seconds (total)`
: Sum of the connect seconds found in the first portion of the breakdown.

`CPU seconds (total)`
: Sum of the CPU seconds found in the first portion of the breakdown.

`Weighted seconds (total)`
: Total weighted CPU seconds; the sum of the weighted seconds found in the first portion of the breakdown.

See Example 3 of the EXAMPLES section for the multitasking breakdown, including summary statistics.

Mark and Disable Options

The mark and disable options are as follows:

`-m` Writes the position of the next accounting record to standard output. This can be used to mark various positions within the job accounting file for later use with the `-p` option. The position marked is the byte offset of the current end-of-information of the job accounting file. (`-m` cannot be used with the report selection and modifier options nor with the `-t` disable option.)

`-t` Disables (terminates) job accounting. (`-m` and `-t` are mutually exclusive).

Report Modifier Options

Report modifier options must be used with at least one selection option. The report modifier options are as follows:

`-d` Provides information about device-specific I/O, if available; forces `-l` to be selected.

`-e` Generates an extended summary report; you must use `-e` with the `-s` option. The following are descriptions of fields produced by specifying the `-e` option with the `-s` option. These fields provide additional accumulated statistics for the reporting period. Several fields contain values only if performance accounting has been enabled; otherwise, the string `NA` is printed instead.

`System Call Time` Total amount of time (in seconds) that the processes executed system calls.

`I/O Wait Time (Terminals)` Total amount of time in seconds that the processes waited for I/O from and to terminals. This field contains a significant value only if performance accounting is enabled (see `devacct`(8)).

`Wait Time while Swapped` Total amount of time (in seconds) that the processes waited while swapped out of memory. This field contains a significant value only if performance accounting is enabled (see `devacct`(8)).

`Number of Swaps` Number of times the processes were swapped out of memory.

`Physical Blocks Moved (Bufd I/O)` Number of physical blocks transferred by processes to and from block devices by using the system buffer I/O interface. This field contains a significant value only if performance accounting is enabled (see `devacct`(8)).

`Physical Blocks Moved (Raw I/O)` Number of physical blocks transferred by processes to and from block devices by using the raw I/O interface. This field contains a significant value only if performance accounting is enabled (see `devacct`(8)).

`-C` Replaces I/O wait times with connect time; must be used with the `-l` option.

`-h` Replaces physical I/O data with the largest amount of memory the process used at one time, in 512-word units. Used only with both the `-c` and `-l` options.

`-l` Provides additional information when used with `-c` or `-f`. Additionally, if the `-l` and `-c` options are used, and there is an MPP accounting record (where there is a Cray MPP system), three MPP fields are printed:

CPU time	Amount of CPU time (in seconds) used by the process for binary execution.
PEs	Number of processing elements (PEs) used.
Barrier bits	Number of barrier bits used to control process flow (in bits per second)

`-r` Raw mode, no headers are printed.

`-a` *acid* Report is for this account ID (*acid*) only.

`-g` *gid* Report is for this group ID (*gid*) only.

`-j` *jid* Report is for this job ID (*jid*) only.

`-u` *uid* Report is for this user ID (*uid*) only.

`-n` *names* Shows only commands matching names patterns that may be regular expressions, as in `ed`(1), except that a + symbol indicates one or more occurrences.

`-p` *marks* Shows only commands within the marked range. This can be a list of ranges with each list item having the following form:

`.`	First command preceding current position
`m1`	First command following mark
`m1:`	All commands between the mark and EOF
`m1:m2`	All commands between the two marks
`:m1`	All commands between BOF and the mark
`:`	All commands between BOF and EOF (default)

See the `-m` option for information on how to obtain marks.

`-D` Reports on tape daemon usage. Tape information, such as the number of bytes read and written, is available after the tape unloads. Reservation information is available after the tape device is released.

NOTES

For multitasking breakdowns with the `-c` and `-s` options, processes are considered to be multitasked if the program was multitasked and if actual execution overlap occurred.

CAUTIONS

In the UNICOS operating system, a system administrator has the option of choosing which accounting records are written to the `pacct` file. Only the base record is required. System administrators may turn off records in the `pacct` file to save disk space and the expense of keeping certain records.

If an accounting record is not turned on in the `pacct` file, a user cannot generate that record's information with the `ja` command. In order to generate the desired information with `ja`, a system administrator must turn on the necessary record(s) in the `pacct` file for a specified time period.

EXAMPLES

The following two examples show the usage of the `-m` and `-p` options with standard shell and Korn shell variables.

Example 1:

```
ja                #enable job accounting
     .
     .  (Miscellaneous commands)
     .
m1=‘ja -m‘        #mark job accounting file’s current position
     .
     .  (Commands of special interest)
     .
m2=‘ja -m‘        #mark job accounting file’s current position
     .
     .  (Miscellaneous commands)
     .
ja -cp $m1:$m2   #print command report from mark m1 to mark m2
ja -st           #print summary report for entire session and disable
                  job accounting
```

Example 2:

```
ja            #enable job accounting
      .
      .   (Miscellaneous commands)
      .
m1='ja -m'    #mark job accounting file's current position
      .
      .   (Commands of special interest)
      .
ja -cp $m1:   #print command report from mark to EOF
      .
      .   (Miscellaneous commands)
      .
ja -st        #print summary report for entire session and disable
               job accounting
```

Example 3: The following example is `ja -s` output for a multitasked process:

```
Job Accounting - Summary Report
===============================

Job Accounting File Name         : jacct
Operating System                 : sn4025 hot 8.0.2ei tja.11 CRAY C90
User Name (ID)                   : user (100)
Group Name (ID)                  : ugrp (10)
Account Name (ID)                : user (100)
Job ID                           : 8271
Report Starts                    : 07/27/94 08:51:58
Report Ends                      : 07/27/94 08:52:10
Elapsed Time                     :           12      Seconds
User CPU Time                    :            3.5826 [      3.4636] Seconds
Multitasking Breakdown

(Concurrent CPUs [Weight] * Connect seconds = CPU seconds [Weighted seconds])
 ---------------  ------    ---------------   -----------  ----------------

               1 [  1.00] *          2.1793 =      2.1793 [          2.1793]
               2 [  1.90] *          0.2464 =      0.4929 [          0.4682]
               3 [  2.70] *          0.2803 =      0.8410 [          0.7569]
               4 [  3.40] *          0.0170 =      0.0679 [          0.0577]

Concurrent CPUs [Weight] * Connect seconds = CPU seconds [Weighted seconds])
     (Avg.)       (Avg.)       (total)         (total)     (total)
 ---------------  ------    --------------   -----------  ----------------

            1.32 [  1.27] *          2.7231 =      3.5811 [          3.4622]
```

```
System CPU Time                 :            0.0812 Seconds
I/O Wait Time (Locked)          :            0.0878 Seconds
I/O Wait Time (Unlocked)        :            0.0337 Seconds
CPU Time Memory Integral        :            1.3551 Mword-seconds
SDS Time Memory Integral        :            0.0000 Mword-seconds
I/O Wait Time Memory Integral   :            0.0424 Mword-seconds
Data Transferred                :            0.0038 MWords
Maximum memory used             :            0.4844 MWords
Logical I/O Requests            :           19
Physical I/O Requests           :           12
Number of Commands              :            2
Billing Units                   :            0.0000
```

SEE ALSO

acctcom(1), at(1), crontab(1), ed(1), ps(1), sh(1)

exec(2), fork(2), listio(2), read(2), reada(2), wait(2), write(2), writea(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

devacct(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

UNICOS Resource Administration, Cray Research publication SG–2302

NAME

jobs – Displays status of jobs in the current session

SYNOPSIS

```
jobs [-l] [-n] [job_id ...]
jobs -p [-n] [job_id ...]
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4
AT&T extension (-n option)

DESCRIPTION

The jobs utility displays the status of jobs that were started in the current shell execution environment (see sh(1)). When jobs reports the termination status of a job, the shell removes its process ID from the list of those "known in the current shell execution environment."

The jobs utility accepts the following options and operand:

-l Lists more information about each job. This information includes the job number, current job, process group ID, state, and the command that formed the job.

-n Displays only jobs that have stopped or exited since last notified.

-p Displays only the process IDs for the process group leaders of the specified jobs.

job_id Specifies the jobs for which the status will be displayed. If no *job_id* operand is given, the status information for all jobs is displayed. See the Jobs subsection in the sh(1) man page for a description of the format of *job_id*.

By default, the jobs utility displays the status of all stopped jobs, running background jobs, and all jobs whose status has changed and have not been reported by the shell.

If you specify the -p option, the output consists of one line for each process ID:

"<*process ID*>\n"

Otherwise, if the -l option is not specified, the output is a series of lines of the form:

"[<*job-number*>] <*current*> <*state*> <*command*>\n"

The fields are as follows:

<*current*> The character + identifies the jobs that would be used as a default for the bg(1) or fg(1) utilities; you can also specify this job using the *job_id* "%+" or "%%". The character - identifies the jobs that would become the default if the current default job were to exit; you can also specify this job using the *job_id* "%-". For other jobs, this field is a <space>. At most, one job can be identified with + and at most one job can be identified with -. If any suspended job exists, the current job is a suspended job. If at least two suspended jobs exist, the previous job also is a suspended job.

<*job-number*> A number that can be used to identify the process group to the wait(1), fg(1), bg(1), and kill(1) utilities. Using these utilities, you can identify the job by prefixing the job number with %.

<*state*> One of the following strings (in the POSIX locale):

Running	Indicates that the job has not been suspended by a signal and has not exited.
Done	Indicates that the job completed and returned exit status zero.
Done (*code*)	Indicates that the job completed normally and that it exited with the specified nonzero exit status, *code*.
Stopped Stopped (SIGTSTP)	Indicates that the job was suspended by the SIGTSTP signal.
Stopped (SIGSTOP)	Indicates that the job was suspended by the SIGSTOP signal.
Stopped (SIGTTIN)	Indicates that the job was suspended by the SIGTTIN signal.
Stopped (SIGTTOU)	Indicates that the job was suspended by the SIGTTOU signal.

<*command*> Specifies the associated command that was given to the shell.

If you specify the -l option, a field that contains the process group ID is inserted before the <*state*> field.

NOTES

The jobs utility described in this man page is a built-in utility to the standard shell (sh(1)). An executable version of this utility is available in /usr/bin/jobs.

EXIT STATUS

The jobs utility exits with one of the following values:

0 Successful completion.

>0 An error occurred.

SEE ALSO

bg(1), fg(1), kill(1), sh(1), wait(1)

NAME

join – Joins specified lines of files

SYNOPSIS

join [-a *file_number* | -v *file_number*] [-e *string*] [-o *list*] [-t *char*] [-1 *field*] [-2 *field*] *file1 file2*

Obsolescent version; may not be supported in future releases:

join [-a *file_number*] [-e *string*] [-j *field*] [-j1 *field*] [-j2 *field*] [-o *list...*] [-t *char*] *file1 file2*

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The join utility joins on standard output the two relations specified by the lines of *file1* and *file2*. If *file1* or *file2* is -, standard input will be used.

file1 and *file2* must be sorted in increasing ASCII-collating sequence on the fields on which they are to be joined, usually the first in each line.

One line is in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line typically consists of the common field, the rest of the line from *file1*, and the rest of the line from *file2*.

The default input field separators are a <space>, <tab>, or <newline> character. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a <space>.

Some of the following options use argument *file_number*. This argument should be 1 or 2, referring to either *file1* or *file2*, respectively. The join utility accepts the following options:

-a *file_number*	In addition to the typical output, produces a line for each unpairable line in file *file_number*, where *file_number* is 1 or 2.
-e *string*	Replaces empty output fields with string *string*.
-j *field*	Equivalent to -1 *field* -2 *field*.
-j1 *field*	Equivalent to -1 *field*.
-j2 *field*	Equivalent to -2 *field*.

`-o` *list*	Each output line comprises the fields specified in *list*, each element of which has the form *n* . *m* (*n* is a file number, and *m* is a field number). Element zero (0) represents the join field. The common field is not printed unless specifically requested. *list* is a single command line argument. However, in the obsolete version, the argument *list* can be multiple arguments on the command line.
`-t` *char*	Uses character *char* as a separator. Every appearance of *char* in a line is significant. Character *char* is used as the field separator for both input and output.
`-v` *file_number*	Instead of the default output, produces a line only for each unpairable line in *file_number*, where *file_number* is 1 or 2. If you specify both `-v 1` and `-v 2`, all unpairable lines are output.
`-1` *field*	Join on the *field*th field of file 1. Fields start with 1.
`-2` *field*	Join on the *field*th field of file 2. Fields start with 1.
file1 file2	The names of the input files that you specify.

NOTES

If this utility is installed with a privilege assignment list (PAL), a user with one of the following active categories is allowed to perform the actions shown:

Active Category	Action
`system`, `secadm`	Allowed to join any files. In a privileged administrator shell environment, shell-redirected I/O is not subject to file protections.
`sysadm`	Allowed to join any files subject to security label restrictions. Shell-redirected I/O is subject to security label restrictions.

If the `PRIV_SU` configuration option is enabled, the super user is allowed to join any files. Shell-redirected I/O on behalf of the super user is not subject to file protections.

EXIT STATUS

The `join` utility exits with one of the following values:

0 All input files were output successfully.

>0 An error occurred.

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`(1), `comm`(1), and `uniq`(1) are incongruous.

File names that are numeric may cause conflict when the `-o` option is used right before listing file names.

EXAMPLES

Example 1: The following command line joins the `passwd` 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 -1 4 -2 3 -o '1.1 2.1 1.6' -t: /etc/passwd /etc/group
```

Example 2: The following command line performs the identical task as the previous example, but using the obsolescent version of `join`:

```
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)

NAME

jstat – Displays job status information

SYNOPSIS

jstat [-j *jid*]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The jstat command displays information pertaining to either one or all active jobs. If the -j option is not specified, all jobs will be displayed with the following format:

```
                nproc      sds      memory    cpu
jid  owner  use lim  use lim  use lim  use lim  command
---  -----  --- ---  --- ---  --- ---  --- ---  -------
```

The fields contain the following information:

jid	Job ID.
owner	ASCII name of job owner.
nproc	Number of processes in use and limit.
sds	Number of blocks in use and limit. A block is 512 Cray words.
memory	Number of clicks in use and limit. A click is 512 Cray words.
cpu	Number of CPU seconds used and limit.
command	Name of current command in job.

**** is used to indicate no limit for all lim fields.

When the -j option is specified, all processes are displayed for the specified job with the following format:

```
pid  status  state  utime  stime  size  addr  system call  command
---  ------  -----  -----  -----  ----  ----  -----------  -------
```

The fields contain the following information:

pid	Process ID.
status	Current status (run or sleep).
state	Current state.
utime	User time in seconds.
stime	System time in seconds.

`size`	Size of process in clicks.
`addr`	Address (in decimal) of process in memory.
`system call`	Last system call made.
`command`	Process name.

NOTES

Output from `jstat` is restricted to processes running at a security label that the calling user dominates.

If this command is installed with the default privilege assignment list (PAL), a user with the `showall` privilege text is not subject to output restrictions.

If *jid* is not a number, zero is used for the *jid*.

You must have the permissions of `/unicos` set to 644 in order for this command to execute correctly. Failure to have permissions set correctly may cause `jstat` to issue the message `permission denied`.

SEE ALSO

`privtext`(1), `ps`(1)

UNICOS Basic Administration Guide for CRAY J90 Model V based Systems, Cray Research publication SG–2416

NAME

kcp – Copies remote files and directories

SYNOPSIS

kcp [-b] [-c *bufsize*] [-p] [-r] [-x] [-k *realm*] [-s *bufsize*] [-S *tos*] *sourcelist destination*

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The kcp utility copies files between machines. The *sourcelist* argument can refer to remote files, local files, or directories; arguments can consist of either absolute or relative path names.

Remote files are specified in the format *rhost*:*file*; *rhost* is a remote host name or alias (described in hosts(5)). When the login name differs from your login name on a Cray Research system, file names on a remote host are specified as *user@rhost*:*file*. If you do not specify a full path name, the path is interpreted relative to your login directory on *rhost*. A path name on a remote host can be quoted (using \, ", or ‘) so that metacharacters are interpreted remotely.

The local file name *file* may not contain a colon (:) unless it is preceded anywhere in the name by a slash (/).

The kcp utility accepts the following options:

-b Displays the buffer sizes for the copy buffer and socket buffer during a transfer.

-c *bufsize* Sets the copy buffer size. This buffer is used for reading and writing between the data socket and the source or destination file. kcp automatically chooses a copy buffer size; however, you can alter the selection. The -c option requires an argument. An argument of off turns off copy buffer sizing and the buffer defaults to the size specified by the COPYBUFSIZE #define in the tcp_config.h file. An argument of auto sets buffer sizing to automatic and has no effect relative to default operation. A numeric argument sets the buffer to that size. The letter K or k can follow a numeric buffer size to specify a multiple of 1024. The default setting is auto. A size of 0 is synonymous with auto.

-p Preserves in its copies the modification times and modes of the source files, ignoring the user file creation mode mask (see umask(1)). By default, the mode and owner of the destination file are preserved if they already existed; otherwise, the mode of the source file modified by the umask on the destination host is used.

-r Copies each subtree that is rooted at that name when any of the source files are directories. The destination must be a directory. If you specify the -r option and any of the source files are directories, kcp copies each subtree rooted at that name; in this case, the destination must be a directory.

`-x` Selects encryption of all information that is transferring between hosts. This option is not available outside the United States and Canada. This option does not work correctly.

`-k` *realm* Specifies that `kcp` must obtain tickets for the remote host in *realm* instead of in the remote host's realm as determined by `krb_realmofhost`(3K).

`-s` *bufsize* Sets the socket buffer size. This kernel buffer is for data transfer in the data socket. `kcp` automatically chooses a socket buffer size; however, you can alter the selection. The `-s` option requires an argument. An argument of `off` turns off socket buffer sizing and the buffer defaults to the default kernel socket buffer size. An argument of `auto` sets buffer sizing to automatic and has no effect relative to default operation. A numeric argument sets the buffer to that size. The letter `K` or `k` can follow a numeric buffer size to specify a multiple of 1024. The default setting is `auto`. A size of 0 is synonymous with `auto`.

`-S` *tos* Sets the IP Type-of-Service (TOS) option for the connection to the value *tos*, which can be a numeric TOS value or a symbolic TOS name that is found in the `/etc/iptos` file.

sourcelist Specifies one or more remote files, local files, or directories.

destination Specifies the destination file or directory.

The `kcp` utility does not prompt for passwords; it uses Kerberos authentication when connecting to *rhost*. Authorization is as described in `klogin`(1).

The `kcp` utility handles third party copies, in which neither source nor target files are on the current machine. Host names also can take the form *rname@rhost* rather than use the current user name on the remote host.

BUGS

When only a directory is legal, the `kcp` utility does not detect all cases in which the target of a copy is a file.

The `kcp` utility is confused by any output that is generated by commands in a `.login`, `.profile`, or `.cshrc` file on the remote host.

When the destination machine is running the 4.2BSD version of `kcp`, you might need to specify the destination user and host name as *rhost.rname*.

Kerberos is used only for the first connection of a third-party copy; the second connection uses the standard Berkeley `rcp` protocol.

The `-x` option does not work with the current MIT code.

SEE ALSO

`cp`(1), `ftp`(1B) `klogin`(1), `rlogin`(1B), `rcp`(1) (UCB version), `rsh`(1), `umask`(1)

`kerberos`(3K), `krb_realmofhost`(3K) in the *Kerberos User's Guide*, Cray Research publication SG–2409

NAME

kdestroy – Destroys Kerberos tickets

SYNOPSIS

kdestroy [-f] [-n] [-q]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The kdestroy utility destroys the user's active Kerberos authorization tickets by writing zeros to the file that contains them. If the ticket file does not exist, kdestroy displays a message accordingly.

After overwriting the file, kdestroy removes the file from the system. The utility displays a message that indicates the success or failure of the operation. If kdestroy cannot destroy the ticket file, the utility warns you by beeping your terminal.

kdestroy also invalidates all Kerberos credentials that are stored in the kernel for the user. These credentials are used for network file system (NFS) requests.

For Kerberos tickets that are obtained from a Cray Research host, you probably will want to place the kdestroy command in your .logout file, so that your tickets can be destroyed automatically when you log out.

The kdestroy utility accepts the following options:

-f Runs without displaying the status message.

-n Keeps valid the NFS credentials that are in the kernel. They remain valid until their normal expiration time. New credentials can only be obtained by executing another kinit(1) command.

-q Does not beep your terminal when it does not destroy the tickets.

BUGS

Only the tickets in the user's current ticket file are destroyed. Separate ticket files are used to hold root instance and password changing tickets. These files must be destroyed also; otherwise, all of a user's tickets must be kept in a single ticket file.

FILES

Tickets are stored in the /tmp/tkt[uid] file unless the user has set the KRBTKFILE environment variable to be another file. Then, the indicated file is used.

SEE ALSO

kinit(1), klist(1)

NAME

keylogin – Decrypts and stores a secret key

SYNOPSIS

keylogin

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The keylogin command informs the secure Remote Procedure Call (RPC) subsystem of your intent to use secure RPC. keylogin communicates with the local keyserv(8) process, allowing it to cache information it will use when performing AUTH_DES style RPC authentication.

The keylogin command prompts you for your secure RPC password, and it uses it to decrypt your secret key stored in the publickey(3R) database. After it is decrypted, your key is stored by the local key server process keyserv(8), to be used by any secure network services, such as NFS.

NOTES

The login command will not call keylogin directly by default; therefore, all users who want to use secure RPC must first explicitly run keylogin.

SEE ALSO

chkey(1), login(1)

Remote Procedure Call (RPC) Reference Manual, Cray Research publication SR–2089

keyserv(8), newkey(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

`kill` – Terminates or signals processes

SYNOPSIS

`kill -s` *signal_name pid...*
`kill -l` [*exit_status*]
`kill -v`

Obsolescent version; may not be supported in future releases:

`kill` [*-signal_name*] *pid...*
`kill` [*-signal_number*] *pid...*

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The `kill` utility sends a signal to the process(es) specified by each *pid* operand. By default, signal number 15 (`SIGTERM`) is sent to the specified process(es). This usually kills processes that do not catch or ignore the signal.

The `kill` utility accepts the following options:

`-l` Causes all values of *signal_name* supported to be written to the standard output, if no *exit_status* operand is specified. If you specify an *exit_status* operand and it has a value of the `?` shell special parameter that corresponds to a process that was terminated by a signal, the *signal_name* corresponding to the signal that terminated the process is written. If you specify an *exit_status* operand and it is the unsigned decimal value of a signal number, the *signal_name* that corresponds to that signal is written.

`-s` *signal_name* Sends signal *signal_name* to the process. *signal_name* is specified as a symbolic name without the `SIG` prefix [see `kill`(2)]. The symbolic name `0` is recognized as representing the signal value 0.

`-v` Causes all signal numbers and their associated *signal_name* to be written to the standard output.

-signal_name Equals `-s` *signal_name*.

-signal_number Specifies a nonnegative decimal integer, *signal_number*, representing the signal to be used instead of `SIGTERM`.

exit_status A decimal integer specifying a signal number or the exit status of a process terminated by a signal.

pid A decimal integer specifying a process or process group to be signaled. The process(es) selected by positive, negative, and zero values of the *pid* operand are the same as described for `kill`(2). If the first *pid* operand is negative, it should be preceded by `--` to keep it from being interpreted as an option.

The process(es) may also be specified as a job control job ID (see `sh`(1)) that identifies a background process group to be signaled. The job control job ID notation is applicable only for invocations of `kill` in the current shell execution environment.

In the obsolescent versions, if the first argument is a negative integer, the argument is interpreted as a *-signal_number* option, not as a negative *pid* operand specifying a process group.

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 process are described in `kill`(2). For example, when process number 0 is specified, all processes in the process group are signaled.

The process to be killed must belong to the current user; only an appropriately authorized user can kill a process owned by another user.

NOTES

If this utility is installed with a privilege assignment list (PAL), a user with one of the following active categories is allowed to perform the actions shown:

Active Category	Action
`system`, `secadm`	Allowed to kill any process.
`sysadm`	Allowed to kill a process owned by another user, subject to security label restrictions. Shell-redirected I/O is subject to security label restrictions.

If the `PRIV_SU` configuration option is enabled, the super user is allowed to kill any process.

The `kill` utility described on this man page is a built-in utility to the standard shell (`sh`(1)). An executable version of this utility is available in `/bin/kill`.

The `csh`(1) utility has a built-in `kill` utility with slightly different characteristics. See `csh`(1).

EXIT STATUS

The `kill` utility exits with one of the following values:

0 At least one matching process was found for each *pid* operand, and the specified signal was successfully processed for at least one matching process.

>0 An error occurred.

EXAMPLES

Example 1: Sends the software termination signal to process ID of 3228:

```
kill 3228
```

Example 2: Same as example 1, but explicitly sends the name of the signal to the process:

```
kill -s TERM 3228
```

Example 3: Determines whether a command terminated due to a signal and the name of signal that caused the command to terminate:

```
utility_name arg1 arg2
kill -l $?
```

Example 4: Sends a SIGQUIT (signal number 3) to two background processes in the current shell execution environment:

```
kill -s QUIT %1 %3
```

SEE ALSO

csh(1), ps(1), sh(1)

kill(2), signal(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

General UNICOS System Administration, Cray Research publication SG–2301

NAME

kinit – Logs in to the Kerberos authentication and authorization system

SYNOPSIS

kinit [-irvl]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The kinit utility logs in to the Kerberos authentication and authorization system. Only registered Kerberos users can use the Kerberos system. For information about registering as a Kerberos user, see the kerberos(7) man page.

When you use kinit without options, the utility prompts for your user name and the Kerberos password and tries to authenticate your login with the local Kerberos server. kinit does not send your Kerberos password across the network in the clear.

If Kerberos authenticates the login attempt, kinit retrieves your initial ticket and puts it in the ticket file that your KRBTKFILE environment variable specified. If this variable is undefined, your ticket is stored in the /tmp directory, in the tkt*uid* file; *uid* specifies your user identification number.

Be sure to use the kdestroy(1) command to destroy any active tickets before you end your login session. You can put the kdestroy(1) command in your .logout file so that your tickets can be destroyed automatically when you log out.

The kinit utility accepts the following options:

-i Directs kinit to prompt you for a Kerberos instance.

-r Directs kinit to prompt you for a Kerberos realm. This option lets you authenticate yourself with a remote Kerberos server. (Interrealm authorization is cumbersome in Kerberos version 4.)

-v (Verbose mode) Directs kinit to print the name of the Kerberos realm, and to issue a status message that indicates the success or failure of your login attempt.

-l Directs kinit to prompt you for a ticket lifetime in minutes. Because of protocol restrictions in Kerberos version 4, this value must be between 5 and 1275 minutes.

BUGS

The -r option is not fully implemented.

SEE ALSO

kdestroy(1), klist(1)

kerberos(7) (available only online)

NAME

klist – Provides list of currently held Kerberos tickets

SYNOPSIS

klist [-s | -t] [-file *name*] [-srvtab]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The klist utility prints the name of the tickets file and the identity of the principal for which the tickets are provided (as listed in the tickets file), and it lists the principal names of all Kerberos tickets currently held by the user, along with the issue and expire time for each authenticator. Principal names are listed in the form *name.instance@realm*, with the dot (.) omitted if the instance is null, and the at sign (@) omitted if the realm is null.

The klist utility accepts the following options:

-s Specifies that klist does not print the issue and expire times, the name of the tickets file, or the identity of the principal.

-t Directs klist to check for the existence of a nonexpired ticket-granting ticket in the ticket file. If one is present, it exits with status 0; otherwise, it exits with status 1. When you specify this option, no output is generated.

-file *name* Specifies *name* as the ticket file. Otherwise, if you set the KRBTKFILE environment variable, this value is used. If you do not set this environment variable, the /tmp/tkt*uid* file is used; *uid* is the current user ID of the user.

-srvtab Specifies that the file is treated as a service key file, and it prints the names of the keys that it contains. If you do not specify the file with a -file option, the default is /etc/srvtab.

BUGS

When a file is being read as a service key file, very little sanity or error checking is performed.

FILES

/etc/krb.conf	File that gets the name of the local realm
/etc/srvtab	Default service key file
/tmp/tkt*uid*	Default ticket file (*uid* is the decimal UID of the user)

SEE ALSO

kdestroy(1), kinit(1)

NAME

`klogin` – Performs remote login

SYNOPSIS

`klogin` *rhost* [`-d`] [`-e`*c*] [`-k` *realm*] [`-l` *username*] [`-x`] [`-8`] [`-E`] [`-K`] [`-L`] [`-S` *tos*]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `klogin` utility connects your terminal on the current local host system to the remote host system *rhost*. It uses Kerberos authentication to determine the authorization to use a remote account. Your remote terminal type is the same as your local terminal type (as specified in your `TERM` environment variable). All echoing occurs at the remote site; therefore, the `klogin` is transparent (except for delays). Flow control through <`CONTROL-s`> and <`CONTROL-q`> and flushing interrupt input and output are handled properly.

Users can create a private authorization list in a `.klogin` file in their login directories. Each line in this file should contain a Kerberos principal name of the form *principal.instance@realm*. *principal* is either an end user or a network service offered by a host computer; *instance* further qualifies the principal. If the principal is a service, the instance specifies the name of the machine on which that service runs. If the principal is a user name that has general user privileges, the instance is usually null.

If the originating user is authenticated to one of the principals specified in `.klogin`, access is granted to the account. If no `.klogin` file exists, the principal *username@localrealm* is granted access; *localrealm* is the name of an administration entity that maintains authentication data.

For example, if `user1` wants `user2` to use `klogin` to log in to `user1`'s account, `user1` must create a `.klogin` file that contains the following:

```
user1@CRAY.COM
user2@CRAY.COM
```

Note that `user1`'s user name must be put in the file; otherwise, access to the account is denied.

If `klogin` encounters a problem obtaining the Kerberos authentication information, it prints an error message and exits.

A line of the form `~.` disconnects from the remote host; `~` is the escape character. A line of the form `~`<`CONTROL-z`> suspends the `klogin` process, and a line of the form `~`<`CONTROL-y`> suspends the send portion of the `klogin` process, but allows output from the remote system. A line of the form `~z` is the same as `~`<`CONTROL-z`>.

The `klogin` utility accepts the following options:

rhost Specifies the name of the remote host.

`-d` Uses `setsockopt`(2) to turn on socket debugging on the TCP sockets that are used for communication with the remote host.

`-e`*c* Specifies an escape character (*c*). No space can separate this option flag (`-e`) and the new escape character (*c*).

`-k` *realm* Directs `klogin` to obtain tickets for the remote host in *realm* instead of in the remote host's realm, as determined by `krb_realmofhost`(3K).

`-l` *username* Specifies the account name (*username*) to use when logging in to the remote machine. The default is the current account name.

`-x` Turns on Data Encryption Standard (DES) encryption for all data passed during the `klogin` session. This option is not available outside the United States and Canada. This option significantly reduces response time and significantly increases CPU use.

`-8` Allows the transmission of 8-bit data.

`-E` Stops any character from being recognized as an escape character. When used with the `-8` option, this provides a completely transparent connection.

`-K` The `-K` option turns off all Kerberos authentication.

`-L` This allows the `klogin` process to be run in `-opost` mode (see `stty`(1)).

`-S` *tos* Sets the IP Type-of-Service (TOS) option for the connection to the value *tos*, which can be a numeric TOS value or a symbolic TOS name found in the `/etc/iptos` file.

BUGS

More of the environment should be propagated.

FILES

`/etc/hosts` TCP/IP host name database

SEE ALSO

`login`(1), `rlogin`(1B), `rsh`(1), `stty`(1)

`setsockopt`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

`kerberos`(3K), `krb_realmofhost`(3K), `krb_sendauth`(3K) in the *Kerberos User's Guide*, Cray Research publication SG–2409

NAME

kpasswd – Changes a user's Kerberos password

SYNOPSIS

kpasswd [-h] [-n *name*] [-i *instance*] [-r *realm*] [-u *username*[.*instance*][@*realm*]]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The kpasswd command changes a Kerberos principal's password.

kpasswd accepts the following options:

-h Prints a brief summary of the options and exits.

-n *name* Uses *name* as the principal name rather than the user name of the user running kpasswd. (*name* is determined from the ticket file, if it exists; otherwise, it is determined from the UNIX user ID.)

-i *instance* Uses *instance* as the instance rather than using a null instance.

-r *realm* Uses *realm* as the realm rather than using the local realm.

-u *username*[.*instance*][@*realm*]
Indicates a fully qualified Kerberos principal.

The command prompts for the current Kerberos password (printing the name of the principal for which it intends to change the password), which is verified by the Kerberos server. If the old password is correct, the user is prompted twice for the new password. A message is printed, indicating the success or failure of the password-changing operation.

BUGS

The kpasswd command does not handle names, instances, or realms that have special characters in them when the -n, -i, or -r options are used. If you specify the -u option, however, any valid user name is accepted.

If the principal does not exist for the password you are trying to change, you will not be told until after you have entered the old password.

The kpasswd command depends on a compatible implementation of kadmind running on the Kerberos server. Such a server is provided in the MIT Project Athena distribution of Kerberos version 4. Incompatible implementations include the kpasswdd program distributed with Berkeley 4.3-Reno.

SEE ALSO

kinit(1), passwd(1)

NAME

krsh – Connects to the remote shell

SYNOPSIS

krsh *host* [-l *username*] [-n] [-d] [-k *realm*] [-S *tos*] [*command*]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The krsh utility connects to the specified host and executes the specified command. krsh copies its standard input to the remote command, copies the standard output of the remote command to its standard output, and copies the standard error of the remote command to its standard error. Interrupt, quit, and terminate signals are propagated to the remote command; krsh usually terminates when the remote command does.

The krsh utility accepts the following options:

host Specifies the name of the host to which krsh will connect.

-l *username* Specifies the remote user name to be used. If you omit this option, your local user name is used. Kerberos authentication is used, and authorization is determined as in klogin(1).

-n Redirects input from the special device /dev/null. See the BUGS section for more information on redirecting input.

-d Turns on socket debugging (through setsockopt(2)) on the TCP sockets that are used for communication with the remote host.

-k *realm* Directs krsh to obtain tickets for the remote host in *realm* instead of in the remote host's realm, as determined by the krb_realmofhost(3K) command.

-S *tos* Sets the IP Type-of-Service (TOS) option for the connection to the value *tos*, which can be a numeric TOS value or a symbolic TOS name that is found in the /etc/iptos file.

command Specifies the command to be executed. If you omit *command*, instead of executing a single command, you will be logged in on the remote host through rlogin(1B).

Shell metacharacters that are not enclosed in quotation marks are interpreted on the local machine; those that are enclosed in quotation marks are interpreted on the remote machine. Thus, in the following example, the first command appends the remote file remotefile to the local file localfile; the second command appends remotefile to otherremotefile:

```
krsh otherhost cat remotefile >> localfile

krsh otherhost cat remotefile ">>" otherremotefile
```

BUGS

If you are using csh(1) and you put a krsh utility in the background without redirecting its input away from the terminal, it will block, even if the remote command posts no read operations. If no input is desired, you should use the -n option to redirect the input of krsh to /dev/null.

You cannot run an interactive command (such as vi(1)); use klogin(1).

Stop signals stop only the local krsh process.

FILES

/etc/hosts	TCP/IP host name database

SEE ALSO

csh(1), klogin(1), login(1), rlogin(1B), vi(1)

setsockopt(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

kerberos(3K), krb_realmofhost(3K), krb_sendauth(3K) in the *Kerberos User's Guide*, Cray Research publication SG–2409

NAME

ksh, rksh, sh, rsh – Korn shell and standard shell, command and programming language

SYNOPSIS

ksh [-a] [-b] [-C] [-c *string*] [-e] [-f] [-h] [-i] [-k] [-m] [-n] [-o *option*] [-p] [-r] [-S] [-s] [-t] [-u] [-v] [-x] [*arg*]

rksh [-a] [-b] [-C] [-c *string*] [-e] [-f] [-h] [-i] [-k] [-m] [-n] [-o *option*] [-p] [-r] [-S] [-s] [-t] [-u] [-v] [-x] [*arg*]

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4
AT&T extensions (-h, -k, -p, -r, and -t options)
CRI extensions (-S option)

DESCRIPTION

sh and rsh invoke the standard shell, which is functionally equivalent to the Korn shell.

ksh is a command interpreter and programming language that executes commands read from a terminal or a file. rksh is a restricted version of the command interpreter ksh; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell.

Invocation

If the shell is invoked by exec(2), and the first character of argument 0 ($0) is -, the shell is assumed to be a login shell and commands are read from /etc/profile and then from either .profile in the current directory or $HOME/.profile, if either file exists. Next, commands are read from the file named by the ENV parameter and parameter substitution is performed, if the file exists. If the -s flag is not present and *arg* is present, a path search is performed on the first *arg* to determine the name of the script to execute. The *arg* script must have read permission and any setuid and setgid settings will be ignored. Commands are then read as described below; the following flags are interpreted by the shell when it is invoked:

-c *string*
If the -c flag is present, commands are read from *string*.

-s If the -s flag is present or no arguments remain, commands are read from the standard input. Shell output, except for the output of the 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 (as told by `ioctl`(2)), this shell is interactive. In this case, `TERM` is ignored (so that `kill 0` does not kill an interactive shell) and `INTR` 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 in the Special Commands subsection.

`rksh` Only

The `rksh` command is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of `rksh` are identical to those of `sh`, except that the following are disallowed:

- Changing directory (see `cd`(1))
- Setting the value of `SHELL`, `ENV`, or `PATH`
- Specifying path or command names containing `/`
- Redirecting output (`>`, `>|`, `<>`, and `>>`)

These restrictions are enforced after the `.profile` and `ENV` files are interpreted.

When a command to be executed is found to be a shell procedure, `rksh` invokes `ksh` to execute it. Thus, it is possible to provide the end user with 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).

Definitions

A metacharacter is one of the following characters:

```
;   &   (   )   |   <   >   <newline>   <space>   <tab>
```

A blank is a `<tab>` or a `<space>`. An identifier is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for functions and named parameters. A word is a sequence of characters separated by one or more metacharacters not enclosed in quotation marks.

A command is a sequence of characters in the syntax of the shell language. The shell reads each command and carries out the desired action either directly or by invoking separate utilities. A special command is a command that is carried out by the shell without creating a separate process. Except for documented side effects, most special commands can be implemented as separate utilities.

Commands

A simple command is a sequence of blank-separated words that may be preceded by a parameter assignment list. See the Environment subsection. 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 (for a list of status values, see `signal`(2)).

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 (see `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 five 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 (that is, the shell does not wait for that pipeline to finish). The symbol `|&` causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell. The standard input and output of the spawned command can be written to and read from by the parent shell by using the `-p` option of the special commands `read` and `print` described later. The symbol `&&` (`||`) causes the list following it to be executed only if the preceding pipeline returns a value of zero for `&&` (nonzero for `||`). An arbitrary number of new lines may appear in a list, instead of a semicolon, to delimit a command.

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` *identifier* [`in` *word* ...] `;do` *list* `;done`

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

`select` *identifier* [`in` *word* ...] `;do` *list* `;done`

A `select` command prints on standard error (file descriptor 2), the set of *word*s, each preceded by a number. If `in` *word* ... is omitted, the positional parameters are used instead (see the Parameter Substitution subsection). The `PS3` prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed *word*s, the value of the parameter *identifier* is set to the *word* corresponding to this number. If this line is empty, the selection list is printed again. Otherwise, the value of the parameter *identifier* is set to null. The contents of the line read from standard input is saved in the parameter `REPLY`. The *list* is executed for each selection until a `break` or end-of-file is encountered.

`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 the File Name Generation subsection).

`if` *list* `;then` *list* [`elif` *list* `;then` *list*] ... [`;else` *list*] `;fi`
The *list* following `if` is executed and, if it returns a 0 exit status, the *list* following the first `then` is executed. Otherwise, the *list* following `elif` is executed and, if its value is 0, 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 0 exit status.

`while` *list* `;do` *list* `;done`
`until` *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 0, executes the `do` *list*; otherwise, the loop terminates. If no commands in the `do` *list* are executed, then the `while` command returns a 0 exit status; `until` may be used in place of `while` to negate the loop termination test.

`(`*list*`)` Execute *list* in a separate environment. If two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below.

`{`*list*`;}` *list* is simply executed. Unlike the metacharacters `(` and `)`, `{` and `}` are reserved words and must come at the beginning of a line or after `;` in order to be recognized.

`[[`*expression*`]]`
Evaluates *expression* and returns a 0 exit status when *expression* is true. For a description of *expression*, see the Conditional Expressions subsection.

`function` *identifier* `{` *list* `;}`
identifier `()` `{` *list* `;}`
Defines a function referenced by *identifier*. The body of the function is the *list* of commands between `{` and `}`. (See the Functions subsection.)

`time` *pipeline*
The *pipeline* is executed and the elapsed time as well as the user and system time are printed on standard error. Because output redirection is set up within `ksh` after the `time` simple command is performed, standard error cannot be redirected to a file.

The following reserved words are recognized only as the first word of a command and when not enclosed in quotation marks:

```
if then else elif fi case esac for while until do done { } function select time [[ ]]
```

Comments

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

Aliasing

The first word of each command is replaced by the text of an `alias` if an `alias` for this word has been defined. The first character of an `alias` name can be any nonspecial printable character, but the rest of the characters must be the same as for a valid *identifier*. The replacement string can contain any valid shell script including the metacharacters listed above. The first word of each command in the replaced text, other than any that are in the process of being replaced, will be tested for aliases. If the last character of the alias value is a blank, the word following the alias will also be checked for alias substitution. Aliases can be used to redefine special built-in commands but cannot be used to redefine the reserved words listed above. Aliases can be created, listed, and exported by using the `alias` command and can be removed by using the `unalias` command. Exported aliases remain in effect for scripts invoked by name, but they must be reinitialized for separate invocations of the shell (see the Invocation subsection).

Aliasing is performed when scripts are read, not while they are executed. Therefore, for an alias to take effect the `alias` definition command must be executed before the command that references the alias is read.

Aliases are frequently used as a short form of full path names. An option to the aliasing facility allows the value of the alias to be set automatically to the full path name of the corresponding command. These aliases are called tracked aliases. The value of a tracked alias is defined the first time the corresponding command is looked up and becomes undefined each time the `PATH` variable is reset. These aliases remain tracked so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell. The `-h` option of the `set` command makes each referenced command name into a tracked alias.

The following exported aliases are compiled into the shell but can be unset or redefined:

```
autoload='typeset -fu'
command='command'
false='let 0'
functions='typeset -f'
hash='alias -t -'
history='fc -l'
integer='typeset -i'
local=typeset
nohup='nohup'
r='fc -e -'
stop='kill -STOP'
suspend='kill -STOP $$'
true=':'
type='whence -v'
```

Tilde Substitution

After alias substitution is performed, each word is checked to see whether it begins with an unquoted tilde (~). If it does, then the word up to a `/` is checked to see whether it matches a user name in the `/etc/passwd` file. If a match is found, the `~` and the matched login name is replaced by the login directory of the matched user. This is called a tilde substitution. If no match is found, the original text is left unchanged. A `~` by itself, or in front of a `/`, is replaced by the value of the `HOME` parameter. A `~` followed by a `+` or `-` is replaced by `$PWD` and `$OLDPWD`, respectively.

In addition, tilde substitution is attempted when the value of a variable assignment parameter begins with a ~.

Command Substitution

The standard output from a command enclosed in parenthesis preceded by a dollar sign (`$()`) or a pair of grave accents (``) may be used as part or all of a word; trailing new lines are removed. In the second (archaic) form, the string between the grave accents is processed for special quoting characters before the command is executed. (See the Quoting subsection). The command substitution `$(cat file)` can be replaced by the equivalent but faster `$(<file)`. Command substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

An arithmetic expression enclosed in double parentheses and preceded by a dollar sign (`$(())`) is replaced by the value of the arithmetic expression within the double parenthesis.

Parameter Substitution

A parameter is an identifier, one or more digits, or any of the characters `*`, `@`, #, `?`, `-`, `$`, and `!`. A named parameter (a parameter denoted by an identifier) has a value and `0` or more attributes. Named parameters can be assigned values and attributes by using the `typeset` special command. The attributes supported by the shell are described later with the `typeset` special command. Exported parameters pass values and attributes to the environment.

The shell supports a one-dimensional array facility. An element of an array parameter is referenced by a subscript. A subscript is denoted by a `[`, followed by an arithmetic expression (see the Arithmetic Evaluation subsection) followed by a `]`. To assign values to an array, use `set -A` *name value* . . . . The value of all subscripts must be in the range of 0 through 1023. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array will be created if necessary. Referencing an array without a subscript is equivalent to referencing the element 0. Since `$array` and `${array[0]}` are equivalent, exporting `array` is equivalent to exporting `array[0]`.

The *value* of a named parameter may also be assigned as follows:

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

If the integer attribute, `-i`, is set for *name*, the *value* is subject to arithmetic evaluation as described below.

Positional parameters (parameters denoted by a number) may be assigned values with the `set` special command. Parameter `$0` is set from argument 0 when the shell is invoked.

The character `$` is used to introduce substitutable *parameters.*

${*parameter*}
The shell reads all the characters from ${ to the matching } as part of the same word, even if it contains braces or metacharacters. Substitues the value, if any, of the parameter. The braces are required when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name or when a named parameter is subscripted. If *parameter* is one or more digits, it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is * or @, all the positional parameters, starting with $1, are substituted (separated by a field separator character). If an array *identifier* with subscript * or @ is used, the value for each of the elements is substituted (separated by a field separator character).

${#*parameter*}
If *parameter* is * or @, substitutes the number of positional parameters. Otherwise, substitutes the length of the value of *parameter*.

${#*identifier*[*]}
Substitutes the number of elements in the array *identifier*.

${*parameter*:-*word*}
If *parameter* is set and is nonnull, substitutes its value; otherwise, substitutes *word*.

${*parameter*:=*word*}
If *parameter* is not set or is null, sets it to *word*; then substitutes the value of the parameter. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
If *parameter* is set and is nonnull, substitutes its value; otherwise, prints *word* and exits from the shell. If *word* is omitted, prints a standard message.

${*parameter*:+*word*}
If *parameter* is set and is nonnull, substitutes *word*; otherwise, substitutes nothing.

${*parameter*#*pattern*}
${*parameter*##*pattern*}
If the shell *pattern* matches the beginning of the value of *parameter*, the value of this substitution is the value of the *parameter* with the matched portion deleted; otherwise, substitutes the value of this *parameter*. The first form deletes the smallest matching pattern; the second form deletes the largest matching pattern.

${*parameter*%*pattern*}
${*parameter*%%*pattern*}
If the shell *pattern* matches the end of the value of *parameter*, the value of this substitution is the value of the *parameter* with the matched part deleted; otherwise, substitutes the value of *parameter*. The first form deletes the smallest matching pattern; the second form deletes the largest matching pattern.

In the preceding paragraph, *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 checks only whether or not *parameter* is set.

The following parameters are set automatically 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 executed command.
$	The process number of this shell.
_	Initially, the value _ is an absolute path name of the shell or script being executed as passed in the *environment*. Subsequently it is assigned the last argument of the previous command. This parameter is not set for commands that are asynchronous. This parameter is also used to hold the name of the matching `MAIL` file when checking for mail.
!	The process number of the last background command invoked.

The following environment variables are set by the shell:

ERRNO	The value of `errno` as set by the most recently failed system call. This value is system-dependent and is intended for debugging purposes.
LINENO	The line number of the current line within the script or function being executed.
OLDPWD	The previous working directory set by the `cd`(1) command.
OPTARG	The value of the last option-argument processed by the `getopts` special command.
OPTIND	The index of the last option-argument processed by the `getopts` special command.
PPID	The process number of the parent of the shell.
PWD	The present working directory set by the `cd`(1) command.
RANDOM	Each time this parameter is referenced, a random integer uniformly distributed between 0 and 32,767 is generated. The sequence of random numbers can be initialized by assigning a numeric value to `RANDOM`.
REPLY	This parameter is set by the `select` statement and by the `read` special command when no arguments are supplied.
SECONDS	Each time this parameter is referenced, the number of seconds since shell invocation is returned. If this parameter is assigned a value, the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

The following environment variables are used by the shell:

CDPATH The search path for the cd(1) command.

COLUMNS If this variable is set, the value is used to define the width of the edit window for the shell edit modes and for printing select lists.

EDITOR If the value of this variable ends in emacs, gmacs, or vi and the VISUAL variable is not set, the corresponding option will be turned on. (See set in the Special Commands subsection.)

ENV If this parameter is set, parameter substitution is performed on the value to generate the path name of the script that will be executed when the shell is invoked. (See the Invocation subsection.) This file is typically used for alias and function definitions.

FCEDIT The default editor name for the fc command.

FPATH The search path for function definitions. This path is searched when a function with the -u attribute is referenced and when a command is not found. If an executable file is found, it is read and executed in the current environment.

IFS Internal field separators, usually <space>, <tab>, and <newline>, that are used to separate command words that result from command or parameter substitution and for separating words with the special command read. The first character of the IFS parameter is used to separate arguments for the "$*" substitution. (See the Quoting subsection.)

HISTFILE If this parameter is set when the shell is invoked, the value is the path name of the file that will be used to store the command history. (See the Command Reentry subsection.)

HISTSIZE If this parameter is set when the shell is invoked, the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is 128.

HOME The default argument (home directory) for the cd(1) command.

LINES If this variable is set, the value is used to determine the column length for printing select lists. select lists will print vertically until about two-thirds of LINES lines are filled.

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 arrival of mail in the specified file.

MAILCHECK This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the MAILPATH or MAIL parameters. The default value is 600 seconds. When the time has elapsed, the shell will check before issuing the next prompt.

MAILPATH A colon (:) -separated list of file names. If this parameter is set, the shell informs the user of any modifications to the specified files that have occurred within the last MAILCHECK seconds. Each file name can be followed by a ? and a message to be printed. The message will undergo parameter substitution with the parameter $_ defined as the name of the file that has changed. The default message is you have mail in $_.

PATH The search path for commands. (See the Execution subsection.) The user may not change PATH if executing under rksh (except in .profile).

PS1 The value of this parameter is expanded for parameter substitution to define the primary prompt string which by default is ''$ ''. The character ! in the primary prompt string is replaced by the *command* number. (See the Command Reentry subsection.)

PS2 Secondary prompt string, by default ''> ''.

PS3 Selection prompt string used within a select loop, by default ''#? ''.

PS4 The value of this parameter is expanded for parameter substitution and precedes each line of an execution trace. If omitted, the execution trace prompt is ''+ ''.

SHELL The path name of the shell is kept in the environment. At invocation, if the base name of this variable matches the pattern *r*sh, the shell becomes restricted.

TMOUT If set to a value greater than 0, the shell will terminate if a command is not entered within the prescribed number of seconds after issuing the PS1 prompt. (The shell can be compiled with a maximum bound for this value that cannot be exceeded.)

VISUAL If the value of this variable ends in emacs, gmacs, or vi, the corresponding option will be turned on. (See the Special Command set.)

The shell gives default values to PATH, PS1, PS2, MAILCHECK, TMOUT, and IFS, while HOME, SHELL, ENV, and MAIL are not set at all by the shell (although HOME is set by login(1)). On some systems MAIL and SHELL are also set by login(1)).

Blank Interpretation

After parameter and command substitution, the results of substitutions are scanned for the 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, unless the -f option has been set, 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 lexicographically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. When a pattern is used for file-name generation, the character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly. In other instances of pattern matching, the / and . are not treated specially.

* 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. A - can be included in the character set by putting it as the first or last character.

A *pattern-list* is a list of one or more patterns separated by each other with a `|`. Composite patterns can be formed with one or more of the following:

`?(`*pattern-list*`)`	Optionally matches any one of the given patterns.
`*(`*pattern-list*`)`	Matches 0 or more occurrences of the given patterns.
`+(`*pattern-list*`)`	Matches one or more occurrences of the given patterns.
`@(`*pattern-list*`)`	Matches exactly one of the given patterns.
`!(`*pattern-list*`)`	Matches anything except one of the given patterns.

Quoting

Each of the metacharacters (see the Definitions subsection) has a special meaning to the shell and causes termination of a word unless quoted. A character may be quoted (that is, made to stand for itself) by preceding it with a `\`. The pair `\<newline>` is ignored. All characters enclosed between a pair of single quotation marks (`''`) are quoted. A single quotation mark cannot appear within single quotation marks. Inside double quotation marks (`""`), parameter and command substitution occurs and `\` quotes the characters `\`, `‘`, `"`, and `$`. The meaning of `$*` and `$@` is identical when not quoted or when used as a parameter assignment value or as a file name. However, when used as a command argument, `$*` is equivalent to `"$1`*d*`$2`*d*`..."`, where *d* is the first character of the `IFS` parameter, whereas `$@` is equivalent to `$1 $2` .... Inside grave accents (`‘‘`) `\` quotes the characters `\`, `‘`, and `$`. If the grave accents occur within double quotation marks, `\` also quotes the character `"`.

The special meaning of reserved words or aliases can be removed by quoting any character of the reserved word. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

An ability to perform integer arithmetic is provided with the special command `let`. Evaluations are performed using long arithmetic. Constants are of the form [*base#*]*n* where *base* is a decimal number between 2 and 36 representing the arithmetic base and *n* is a number in that base. If *base* is omitted, base 10 is used.

An arithmetic expression uses the same syntax, precedence, and associativity of expression of the C language. All the integral operators, other than `++`, `--`, `?:`, and `,` are supported. Named parameters can be referenced by name within an arithmetic expression without using the parameter substitution syntax. When a named parameter is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a named parameter can be specified with the `-i` option of the `typeset` special command. Arithmetic evaluation is performed on the value of each assignment to a named parameter with the `-i` attribute. If you do not specify an arithmetic base, the first assignment to the parameter determines the arithmetic base. This base is used when parameter substitution occurs.

Because many of the arithmetic operators require quotation marks, an alternative form of the `let` command is provided. For any command which begins with a `((`, all the characters until a matching `))` are treated as a expression enclosed in quotation marks. More precisely, `(( ... ))` is equivalent to `let "..."`.

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 (that is, the value of PS2) is issued.

Conditional Expressions

A conditional expression is used with the [[compound command to test attributes of files and to compare strings. Word splitting and file-name generation are not performed on the words between [[and]]. Each expression can be constructed from one or more of the following unary or binary expressions:

-a *file*	True, if *file* exists.
-b *file*	True, if *file* exists and is a block special file.
-c *file*	True, if *file* exists and is a character special file.
-d *file*	True, if *file* exists and is a directory.
-e *file*	True, if *file* exists.
-f *file*	True, if *file* exists and is an ordinary file.
-g *file*	True, if *file* exists and is has its setgid bit set.
-h *file*	True, if *file* exists and is a symbolic link.
-k *file*	True, if *file* exists and is has its sticky bit set.
-m *file*	True, if *file* exists and is migrated (type IFOFL).
-M *file*	True, if *file* exists and is migrated (has a DMF handle).
-n *string*	True, if length of *string* is nonzero.
-o *option*	True, if option named *option* is on.
-p *file*	True, if *file* exists and is a fifo special file or a pipe.
-r *file*	True, if *file* exists and is readable by current process.
-s *file*	True, if *file* exists and has size greater than 0.
-t *fildes*	True, if file descriptor number *fildes* is open and associated with a terminal device.
-u *file*	True, if *file* exists and is has its setuid bit set.
-w *file*	True, if *file* exists and is writable by current process.
-x *file*	True, if *file* exists and is executable by current process. If *file* exists and is a directory, the current process has permission to search in the directory.
-z *string*	True, if length of *string* is 0.
-L *file*	True, if *file* exists and is a symbolic link.
-O *file*	True, if *file* exists and is owned by the effective user id of this process.
-G *file*	True, if *file* exists and its group matches the effective group id of this process.
-S *file*	True, if *file* exists and is a socket.
file1 -nt *file2*	True, if *file1* exists and is newer than *file2*.
file1 -ot *file2*	True, if *file1* exists and is older than *file2*.
file1 -ef *file2*	True, if *file1* and *file2* exist and refer to the same file.
string = *pattern*	True, if *string* matches *pattern*.
string != *pattern*	True, if *string* does not match *pattern*.
string1 < *string2*,	True, if *string1* comes before *string2* based on the ASCII value of their characters.
string1 > *string2*	True, if *string1* comes after *string2* based on the ASCII value of their characters.
exp1 -eq *exp2*	True, if *exp1* is equal to *exp2*.

exp1 `-ne` *exp2*	True, if *exp1* is not equal to *exp2*.
exp1 `-lt` *exp2*	True, if *exp1* is less than *exp2*.
exp1 `-gt` *exp2*	True, if *exp1* is greater than *exp2*.
exp1 `-le` *exp2*	True, if *exp1* is less than or equal to *exp2*.
exp1 `-ge` *exp2*	True, if *exp1* is greater than or equal to *exp2*.

In each of the preceding expressions, if *file* is of the form `/dev/fd/`*n*, where *n* is an integer, the test applied to the open file whose descriptor number is *n*.

A compound expression can be constructed from these primitives by using any of the following, listed in decreasing order of precedence:

(*expression*)	True, if *expression* is true. Used to group expressions.
! *expression*	True if *expression* is false.
expression1 && *expression2*	True, if *expression1* and *expression2* are both true.
expression1 || *expression2*	True, if either *expression1* or *expression2* is true.

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 they are not passed on to the invoked command. Command and parameter substitution occurs before *word* or *digit* is used except as noted. File-name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

<*word*	Uses file *word* as standard input (file descriptor 0).
>*word*	Uses file *word* as standard output (file descriptor 1). If the file does not exist, it is created. If the file exists and the `noclobber` option is on, this causes an error; otherwise, it is truncated to 0 length.
>|*word*	Same as >, except that it overrides the `noclobber` option.
>>*word*	Uses 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*	Opens file *word* for reading and writing as standard input.
<<[-]*word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. No parameter substitution, command substitution, or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is enclosed in quotation marks, no interpretation is placed on the characters of the document; otherwise, parameter and command substitution occurs, `\<newline>` is ignored, and \ must be used to quote the characters \, $, ‘, and the first character of *word*. If - is appended to <<, all leading `<tab>`s 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 >& *digit*.
<&-	The standard input is closed. Similarly for the standard output using >&-.

`<&p` The input from the co-process is moved to standard input.

`>&p` The output to the co-process is moved to standard output.

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

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

means that file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (*file descriptor*, *file*) association at the time of evaluation. For example:

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

first associates file descriptor 1 with file *fname*. It then associates file descriptor 2 with the file associated with file descriptor 1 (that is, *fname*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be associated with file *fname*.

If a command is followed by `&` and job control is not active, 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.

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 names must be identifiers and the values are character strings. 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 and marking it export. Executed commands inherit the environment. If the user modifies the values of these parameters or creates new ones, using the `export` or `typeset -x` commands, the commands become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions that must be noted in `export` or `typeset -x` commands.

The environment for any simple command or function may be augmented by prefixing it with one or more parameter assignments. A parameter assignment argument is a word of the form *identifier=value*. As far as the above execution of `cmd` is concerned, the following two lines are equivalent:

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

If the `-k` flag is set, all parameter assignment 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
```

This feature is intended for use with scripts written for early versions of the shell, and its use in new scripts is strongly discouraged, because support will be discontinued in future releases.

Functions

The `function` reserved word, described in the Commands subsection, is used to define shell functions. Shell functions are read in and stored internally. Alias names are resolved when the function is read. Functions are executed like commands with the arguments passed as positional parameters (see the Execution subsection).

Functions execute in the same process as does the caller and share all files and present working directory with the caller. Traps caught by the caller are reset to their default action inside the function. A trap condition that is not caught or ignored by the function causes the function to terminate and the condition to be passed on to the caller. A trap on `EXIT` set inside a function is executed after the function completes in the environment of the caller. Ordinarily, variables are shared between the calling program and the function. However, the `typeset` special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command `return` is used to return from function calls. Errors within functions return control to the caller.

Function identifiers can be listed with the `-f` or `+f` option of the `typeset` special command. The text of functions will also be listed with `-f`. Function can be undefined with the `-f` option of the `unset` special command.

Ordinarily, functions are unset when the shell executes a shell script. The `-xf` option of the `typeset` command allows a function to be exported to scripts that are executed without a separate invocation of the shell. Functions that need to be defined across separate invocations of the shell should be specified in the `ENV` file with the `-xf` option of `typeset`.

Jobs

If the `monitor` option of the `set` command is turned on, an interactive shell associates a job with each pipeline. It keeps a table of current jobs, printed by the `jobs` command, and assigns them small integer numbers. When a job is started asynchronously with &, the shell prints a line that looks as follows:

```
[1] 1234
```

This indicates that the job that was started asynchronously was job number 1 and had one (top-level) process, whose process ID was `1234`.

If you are running a job and want to do something else, you may press `<^z>` (`<CONTROL-z>`), which sends a `STOP` signal to the current job. The shell will then usually indicate that the job has been stopped, and print another prompt. You can then manipulate the state of this job, putting it in the background with the `bg` command, or run some other commands and then eventually bring the job back into the foreground with the foreground command `fg`. The consequence of pressing `<^z>` takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are usually allowed to produce output, but this can be disabled by entering the command `stty tostop`. If you set this `tty` option, background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. A job can be referred to by the process ID of any process of the job or by one of the following:

%*number*	The job with the given number
%*string*	Any job whose command line begins with *string*
%?*string*	Any job whose command line contains *string*
%%	Current job
%+	Equivalent to %%
%-	Previous job

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

When the monitor mode is on, each background job that completes triggers any trap set for `CHLD`.

If you try to leave the shell while jobs are running or stopped, you will be warned that `You have stopped(running) jobs.` You may use the `jobs` command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

Signals

The `INT` and `QUIT` signals for an invoked command are ignored if the command is followed by & and job `monitor` option is not active. Otherwise, signals have the values inherited by the shell from its parent (but see also the `trap` command).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the commands listed in the Special Commands subsection, it is executed within the current shell process. Next, the command name is checked to see whether or not it matches one of the user-defined functions. If it matches, the positional parameters are saved and then reset to the arguments of the `function` call. When the function completes or issues a `return`, the positional parameter list is restored and any trap set on `EXIT` within the function is executed. The value of a function is the value of the last command executed. A function is also executed in the current shell process. If a command name is not a special command or a user-defined function, a process is created and an attempt is made to execute the command by using `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:/usr/ucb (specifying /bin, /usr/bin, /usr/ucb, and the current directory, in that order). The current directory can be specified by two or more adjacent colons, or by a colon at the beginning or end of the path list. If the command name contains a /, the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an a.out file, or if the file does not begin with characters #! (see exec(2)), it is assumed to be a file containing shell commands. A subshell is spawned to read it. All nonexported aliases, functions, and named parameters are removed in this case. The shell command file must have read permission and any setuid and setgid settings will be ignored. A parenthesized command is executed in a subshell without removing nonexported quantities.

Command Reentry

The text of the last HISTSIZE (default 128) commands entered from a terminal device is saved in a history file. The file $HOME/.sh_history is used if the HISTFILE variable is not set or is not writable. A shell can access the commands of all interactive shells that use the same named HISTFILE. The special command fc is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number, or you can specify the first character or characters of the command. A single command or range of commands can be specified. If you do not specify an editor program as an argument to fc, the value of the parameter FCEDIT is used. If FCEDIT is not defined, /bin/ed is used. The edited command(s) is printed and reexecuted upon leaving the editor. The editor name - is used to skip the editing phase and to reexecute the command. In this case, a substitution parameter of the form *old=new* can be used to modify the command before execution. For example, if r is aliased to 'fc -e -' then typing r bad=good c will reexecute the most recent command that starts with the letter c, replacing the first occurrence of the string bad with the string good.

Inline Editing Options

Usually, each command line entered from a terminal device is simply typed followed by a newline (<RETURN> or <LINE FEED>). If either the emacs, gmacs, or vi option is active, the user can edit the command line. To be in either of these edit modes, set the corresponding option. An editing option is automatically selected each time the VISUAL or EDITOR variable is assigned a value ending in either of these option names.

The editing features require that the user's terminal accept <RETURN> as carriage return without line feed and that a space ( ) must overwrite the current character on the screen. ADM terminal users should set the "space - advance" switch to 'space'. Hewlett-Packard series 2621 terminal users should set the straps to bcGHxZ etX.

The editing modes implement a concept in which the user is looking through a window at the current line. The window width is the value of COLUMNS, if it is defined; otherwise the width is 80. The window height is set at the value of LINES, if it is defined; otherwise the height is 24. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user. As the cursor moves and reaches the window boundaries, the window will be centered about the cursor. The mark is a > (<, *) if the line extends on the right (left, both) side(s) of the window.

The search commands in each edit mode provide access to the history file. Only strings are matched, not patterns, although a leading ^ in the string restricts the match to begin at the first character in the line.

Emacs Editing Mode

This mode is entered by enabling either the emacs or gmacs option. The only difference between these two modes is the way they handle <^T>. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All the editing commands are control characters or escape sequences. The notation for control characters is caret (^) followed by the character. For example, <^F> is the notation for pressing <CONTROL-f>. This is entered by pressing the f key while holding down the <CONTROL> key. The <SHIFT> key is not pressed. (The notation ^? indicates the <DEL> (delete) key.)

The notation for escape sequences is M- followed by a character. For example, M-f (pronounced Meta f) is entered by depressing ESC (ASCII 033) followed by f. (M-F would be the notation for ESC followed by SHIFT (capital) F.)

All edit commands operate from anyplace on the line (not just the beginning). Neither the <RETURN> nor the <LINE FEED> key is pressed after edit commands except when noted.

^F	Moves cursor forward (right) 1 character.
M-f	Moves cursor forward 1 word. (To the emacs editor, a *word* is a string of characters consisting of only letters, digits, and underscores.)
^B	Moves cursor backward (left) 1 character.
M-b	Moves cursor backward 1 word.
^A	Moves cursor to start-of-line.
^E	Moves cursor to end-of-line.
^]*char*	Moves cursor forward to character *char* on current line.
M-^]*char*	Moves cursor back to character *char* on current line.
^X^X	Interchanges cursor and mark.
erase	(User-defined erase character as defined by the stty(1) command, usually ^H or #.) Deletes previous character.
^D	Deletes current character.
M-d	Deletes current word.
M-^H	(<Meta-BACKSPACE>) Deletes previous word.
M-h	Deletes previous word.
M-^?	(<Meta-DEL>) Deletes previous word. (If your interrupt character is ^? (<DEL>, the default), this command does not work.)
^T	Transposes current character with next character in emacs mode. Transposes two previous characters in gmacs mode.

^C	Capitalizes current character.
M-c	Capitalizes current word.
M-l	Changes the current word to lowercase.
^K	Deletes from cursor to end-of-line. If preceded by a numerical parameter whose value is less than the current cursor position, deletes from given position up to the cursor. If preceded by a numerical parameter whose value is greater than the current cursor position, deletes from cursor up to given cursor position.
^W	Kills from the cursor to the mark.
M-p	Pushes the region from the cursor to the mark on the stack.
kill	(User-defined kill character as defined by the stty command, usually ^G or @.) Kills the entire current line. If two *kill* characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).
^Y	Restores last item removed from line. (Yanks item back to the line.)
^L	Line feed and prints current line.
^@	(Null character) Sets mark.
M-<*space*>	(<Meta SPACE>) Sets mark.
^J	(New line) Executes the current line.
^M	(<RETURN>) Executes the current line.
eof	End-of-file character, usually ^D, is processed as an end-of-file only if the current line is null.
^P	Fetches previous command. Each time ^P is entered, the previous command back in time is accessed. Moves back one line when not on the first line of a multiline command.
M-<	Fetches the least recent (oldest) history line.
M->	Fetches the most recent (youngest) history line.
^N	Fetches next command line. Each time ^N is entered, the next command line forward in time is accessed.
^R*string*	Reverses search history for a previous command line containing *string*. If a parameter of 0 is given, the search is forward. *String* is terminated by a RETURN or NEW LINE. If string is preceded by a ^, the matched line must begin with *string*. If *string* is omitted, the next command line containing the most recent *string* is accessed. In this case a parameter of 0 reverses the direction of the search.
^O	(Operates) Executes the current line and fetches the next line relative to current line from the history file.
M-*digits*	(Escape) Defines numeric parameter; the digits are taken as a parameter to the next command. The commands that accept a parameter are ^F, ^B, *erase*, ^C, ^D, ^K, ^R, ^P, ^N, ^], M-., M-^], M-_, M-b, M-c, M-d, M-f, M-h, M-l, and M-^H.

M-*letter*	(Soft-key) Your alias list is searched for an alias by the name _*letter* and if an alias of this name is defined, its value will be inserted on the input queue. The *letter* must not be one of the previous meta-functions.
M-[*letter*	(Soft-key) Your alias list is searched for an alias by the name __*letter* and if an alias of this name is defined, its value will be inserted on the input queue. This can be used to program functions keys on many terminals.
M-.	Inserts the last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.
M-_	Same as M-..
M-*	Attempts file-name generation on the current word. An asterisk is appended if the word does not match any file or contain any special pattern characters.
M-ESC	File name completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.
M-=	Lists files matching current word pattern if an asterisk were appended.
^U	Multiplies parameter of next command by 4.
\	Escapes next character. Editing characters, the user's erase, kill and interrupt (usually ^?) characters may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).
^V	Displays version of the shell.
M-#	Inserts a # at the beginning of the line and executes it. This causes a comment to be inserted in the history file.

The vi Editing Mode

The vi editor has two typing modes. When you enter a command, you are in input mode. To edit, you enter control mode by typing ESC (033), move the cursor to the point needing correction, and then insert or delete characters or words as needed. Most control commands accept an optional repeat count prior to the command.

In vi mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command, and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

If the option `viraw` is also set, the terminal will always have canonical processing disabled. This mode may be helpful for certain terminals and is implicit for systems that do not support two alternative end-of-line delimiters (as defined by RFC-1184). If the shell can determine that this functionality is supported, the `viraw` option will be off by default. Otherwise, the `viraw` option will be on by default. If the shell can determine that this functionality is not supported, efforts to turn the `viraw` option off will be silently ignored.

Input Edit Commands

By default, the editor is in input mode.

erase	(User-defined erase character as defined by the `stty` command, usually `^H` or #.) Deletes previous character.
`^W`	Deletes the previous blank separated word.
`^D`	Terminates the shell.
`^V`	Escapes next character. Editing characters, the user's erase or kill characters may be entered in a command line or in a search string if preceded by a `^V`. The `^V` removes the next character's editing features (if any).
`\`	Escapes the next erase or kill character.

Motion Edit Commands

The following commands move the cursor:

[*count*]`l`	Moves cursor forward (right) 1 character.
[*count*]`w`	Moves cursor forward 1 alphanumeric word.
[*count*]`W`	Moves cursor to the beginning of the next word that follows a blank.
[*count*]`e`	Moves cursor to end-of-word.
[*count*]`E`	Moves cursor to end of the current blank delimited word.
[*count*]`h`	Moves cursor backward (left) 1 character.
[*count*]`b`	Moves cursor backward 1 word.
[*count*]`B`	Moves cursor to preceding blank separated word.
[*count*]`|`	Moves cursor to column *count*.
[*count*]`f`*c*	Finds next character *c* in the current line.
[*count*]`F`*c*	Finds previous character *c* in the current line.
[*count*]`t`*c*	Equivalent to `f` followed by `h`.
[*count*]`T`*c*	Equivalent to `F` followed by `l`.
[*count*]`;`	Repeats *count* times, the last single character find command, `f`, `F`, `t`, or `T`.
[*count*]`,`	Reverses the last single character find command *count* times.
`0`	Moves cursor to start of line.
`^`	Moves cursor to first nonblank character in line.
`$`	Moves cursor to end-of-line.

Search Edit Commands

The following commands access your command history:

[*count*]k	Fetches previous command. Each time k is entered, the previous command back in time is accessed.
[*count*]-	Equivalent to k.
[*count*]j	Fetches next command. Each time j is entered, the next command forward in time is accessed.
[*count*]+	Equivalent to j.
[*count*]G	Fetches the command number *count*. The default is the least recent history command.
/*string*	Searches backward through history for a previous command containing *string*. *string* is terminated by a <RETURN> or <NEWLINE>. If *string* is preceded by a ^, the matched line must begin with *string*. If *string* is null, the previous string will be used.
?*string*	Same as / except that search will be in the forward direction.
n	Searches for next match of the last pattern to / or ? commands.
N	Searches for next match of the last pattern to / or ?, but in reverse direction. Searches history for the string entered by the previous / command.

Text Modification Edit Commands

The following commands modify the line:

a	Enters input mode and enters text after the current character.
A	Appends text to the end of the line. Equivalent to $a.
[*count*]c*motion* c[*count*]*motion*	Deletes current character through the character to which *motion* would move the cursor and enter input mode. If *motion* is c, the entire line will be deleted and input mode entered.
C	Deletes the current character through the end-of-line and enter input mode. Equivalent to c$.
S	Equivalent to cc.
D	Deletes the current character through the end-of-line. Equivalent to d$.
[*count*]d*motion* d[*count*]*motion*	Deletes current character through the character to which *motion* would move. If *motion* is d, the entire line will be deleted.
i	Enters input mode and inserts text before the current character.
I	Inserts text before the beginning of the line. Equivalent to 0i.
[*count*]P	Places the previous text modification before the cursor.

[*count*]p	Places the previous text modification after the cursor.
R	Enters input mode and replaces characters on the screen with characters you type overlay fashion.
[*count*]r*c*	Replaces the *count* character(s) starting at the current cursor position with *c*, and advances the cursor.
[*count*]x	Deletes current character.
[*count*]X	Deletes preceding character.
[*count*].	Repeats the previous text modification command.
[*count*]~	Inverts the case of the *count* character(s) starting at the current cursor position and advances the cursor.
[*count*]_	Causes the *count* word of the previous command to be appended and input mode entered. The last word is used if *count* is omitted.
*	Causes an * to be appended to the current word and file-name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.
\	File name completion. Replaces the current word with the longest common prefix of all file names matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.

Other Edit Commands

Miscellaneous commands.

[*count*]y*motion* y[*count*]*motion*	Yanks current character through character that *motion* would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.
Y	Yanks from current position to end-of-line. Equivalent to y$.
u	Undoes the last text modifying command.
U	Undoes all the text modifying commands performed on the line.
[*count*]v	Returns the command `fc -e ${VISUAL:-${EDITOR:-vi}}` *count* in the input buffer. If *count* is omitted, the current line is used.
^L	Line feed and prints current line. Has effect only in control mode.
^J	(<NEWLINE>) Executes the current line, regardless of mode.
^M	(<RETURN>) Executes the current line, regardless of mode.
#	Sends the line after inserting a # in front of the line. Useful for causing the current line to be inserted in the history without being executed.
=	Lists the file names that match the current word if an asterisk were appended to it.

@letter Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value will be inserted on the input queue for processing.

Special Commands

The following simple commands are executed in the shell process. Input/output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1 and the exit status, when there is no syntax error, is 0. Commands that are preceded by one or two † are treated specially in the following ways:

1. Parameter assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after parameter assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a parameter assignment, are expanded with the same rules as a parameter assignment. This means that tilde substitution is performed after the = sign and word splitting and file name generation are not performed.

The special commands are as follows:

† `:` [*arg* ...] The command only expands parameters.

† `.` *file* [*arg* ...] Reads the complete *file* and then executes the commands. The commands are executed in the current shell environment. The search path specified by `PATH` is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise, the positional parameters are unchanged. The exit status is the exit status of the last command executed.

†† `alias` [`-tx`] [*name*[=*value*]] ...
Alias with no arguments prints the list of aliases in the form *name=value* on standard output. An alias is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The `-t` flag is used to set and list tracked aliases. The value of a tracked alias is the full path name corresponding to the given *name*. The value becomes undefined when the value of `PATH` is reset but the aliases remained tracked. Without the `-t` flag, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The `-x` flag is used to set or print exported aliases. An exported alias is defined for scripts invoked by name. The exit status is nonzero if a *name* is given, but no value, for which no alias has been defined. See `alias`(1).

`bg` [*job*...] This command is only on systems that support job control. Puts each specified *job* into the background. The current job is put in the background if *job* is not specified. See the Jobs subsection for a description of the format of `job`. See `bg`(1).

† `break` [*n*] Exits from the enclosing `for`, `while`, `until`, or `select` loop, if any. If *n* is specified, it breaks *n* levels.

`cd` [*arg*]
`cd` *old new*
: This command can be in either of two forms. In the first form, it changes the current directory to *arg*. If *arg* is `-` , the directory is changed to the previous directory. The shell parameter `HOME` is the default *arg*. The parameter `PWD` is set to the current directory. 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). 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 second form of `cd` substitutes the string *new* for the string *old* in the current directory name, `PWD`, and tries to change to this new directory.

  The `cd` command may not be executed by `rksh`. See `cd`(1).

`command` [`-pVv`] *command_name* [*argument* ...]
: See `command`(1) for information on using this command.

† `continue` [*n*]
: Resumes the next iteration of the enclosing `for`, `while`, `until`, or `select` loop. If *n* is specified, it resumes at the *n*th enclosing loop.

`dmmode` *n*
: Sets the data migration recall mode to *n*. For usage and description, see `dmmode`(1).

`echo` [*arg* ...]
: For usage and description, see `echo`(1).

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

† `exec` [*arg* ...]
: If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given, the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 that are opened with this mechanism are closed when invoking another program.

† `exit` [*n*]
: Causes the 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 except for a shell that has the `ignoreeof` option (see `set`) turned on.

†† `export` [*name*[=*value*]] ...
: The specified *name*s are marked for automatic export to the `environment` of subsequently executed commands.

`fc` [`-e` *ename*] [`-nlr`] [*first* [*last*]]

`fc -e -` [*old=new*] [*command*]

In the first form, a range of commands from *first* to *last* is selected from the last `HISTSIZE` commands that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the flag `-l` is selected, the commands are listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, the value of the parameter `FCEDIT` (default `/bin/ed`) is used as the editor. When editing is complete, the edited command(s) is executed. If *last* is not specified, it will be set to *first*. If *first* is not specified, the default is the previous command for editing and –16 for listing. The flag `-r` reverses the order of the commands and the flag `-n` suppresses command numbers when listing.

In the second form, the *command* is reexecuted after the substitution *old=new* is performed. See `fc`(1).

`fg` [*job...*] This command is only on systems that support job control. Each *job* specified is brought to the foreground. Otherwise, the current job is brought into the foreground. For a description of the format of `job`, see the Jobs subsection. See `fg`(1).

`getopts` *optstring name* [*arg* ...]

Checks *arg* for legal options. If *arg* is omitted, the positional parameters are used. An option-argument begins with a `+` or a `-`. An option not beginning with `+` or `-` or the argument `--` ends the options. *optstring* contains the letters that `getopts` recognizes. If a letter is followed by a `:`, that option is expected to have an argument. The options can be separated from the argument by blanks.

`getopts` places the next option letter it finds inside variable *name* each time it is invoked with a `+` prepended when *arg* begins with a `+`. The index of the next *arg* is stored in `OPTIND`. The option-argument, if any, is stored in `OPTARG`.

A leading `:` in *optstring* causes `getopts` to store the letter of an invalid option in `OPTARG`, and to set *name* to `?` for an unknown option and to `:` when a required option is missing. Otherwise, `getopts` prints an error message. The exit status is nonzero when there are no more options. See `getopts`(1).

`jobs` [`-lnp`] [*job* ...]

Lists information about each given job, or all active jobs if *job* is omitted. The `-l` flag lists process IDs in addition to the normal information. The `-n` flag displays only jobs that have stopped or exited since last notified. The `-p` flag causes only the process group to be listed. For a description of the format of *job*, see the Jobs subsection. See `jobs`(1).

`kill -s` *signal_name pid...*
`kill -l` [*exit_status*]
`kill -v`

`kill` [-*signal_name*] *pid...*
`kill` [-*signal_number*] *pid...*
For usage and description, see `kill`(1).

`let` *arg* ... Each *arg* is a separate arithmetic expression to be evaluated. For a description of arithmetic expression evaluation, see the Arithmetic Evaluation subsection.

If the value of the last expression is nonzero, the exit status is `0`; otherwise it is `1`.

`print` [`-Rnprsu`[*n*]] [*arg* ...]
The shell output mechanism. With no flags or with flag `-` or `--`, the arguments are printed on standard output as described by `echo`(1). In raw mode, `-R` or `-r`, the escape conventions of `echo` are ignored. The `-R` option will print all subsequent arguments and options other than `-n`. The `-p` option causes the arguments to be written onto the pipe of the process spawned with `|&` instead of standard output. The `-s` option causes the arguments to be written onto the history file instead of standard output. The `-u` flag can be used to specify a one-digit file descriptor unit number `n` on which the output will be placed. The default is 1. If the flag `-n` is used, no `<newline>` is added to the output.

`pwd` Prints the working directory. Equivalent to `print -r - $PWD`

`read` [`-prsu`[*n*]] [*name*`?`*prompt*] [*name* ...]
The shell input mechanism. One line is read and is broken up into fields, using the characters in `IFS` as separators. In raw mode, `-r`, a `\` at the end of a line does not signify line continuation. The first field is assigned to the first *name*, the second field to the second *name*, and so on, with leftover fields assigned to the last *name*. The `-p` option causes the input line to be taken from the input pipe of a process spawned by the shell using `|&`. If the `-s` flag is present, the input will be saved as a command in the history file. The `-u` flag can be used to specify a one-digit file descriptor unit from which to read. The file descriptor can be opened with the `exec` special command. The default value of *n* is 0. If *name* is omitted, `REPLY` is used as the default *name*. The exit status is 0, unless an end-of-file is encountered. See `read`(1). An end-of-file with the `-p` option causes cleanup for this process so that another can be spawned. If the first argument contains a `?`, the remainder of this word is used as a *prompt* on standard error when the shell is interactive. The exit status is 0, unless an end-of-file is encountered.

†† `readonly` [*name*[`=`*value*]] ...
The specified *names* are marked read only, and these names cannot be changed by subsequent assignment.

† `return` [*n*] Causes a shell *function* to return to the invoking script with the return status specified by *n*. If *n* is omitted, the return status is that of the last command executed. If `return` is invoked while not in a *function* or a `.` script, it is the same as an `exit`.

`set` [±`abCefhkmnopsStuvx`] [±`o` *option*]... [±`A` *name*] [*arg* ...]
Sets options for the shell. The flags for this command have the following meanings:

`-A` Array assignment. Unsets the variable *name* and assigns values sequentially from the list *arg*. If `+A` is used, the variable *name* is not unset first.

`-a` Automatically exports all subsequent defined parameters.

`-b` Causes the shell to notify the user asynchronously of background job completion.

`-C` Prevents redirection `>` from truncating existing files. Requires `>|` to truncate a file when turned on.

`-e` Executes the `ERR` trap, if set, and exits. Used with a command that has a nonzero exit status. This mode is disabled while reading profiles.

`-f` Disables file name generation.

`-h` Each command becomes a tracked alias when first encountered.

`-k` Places all parameter assignment arguments in the command environment. All parameter assignment arguments are placed in the environment for a command, not just those that precede the command name.

`-m` Runs background jobs in a separate process group. A line will print upon completion. The exit status of background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells.

`-n` Reads commands and checks them for syntax errors, but does not execute them. Ignored for interactive shells.

`-o` The following argument can be one of the following option names:

`allexport`	Same as `-a`.
`errexit`	Same as `-e`.
`bgnice`	All background jobs are run at a lower priority. This is the default mode.
`emacs`	Puts you in an `emacs`-style inline editor for command entry.
`gmacs`	Puts you in a `gmacs`-style inline editor for command entry.
`ignoreeof`	The shell will not exit on end-of-file. The command `exit` must be used.
`keyword`	Same as `-k`.
`markdirs`	All directory names resulting from file name generation have a trailing `/` appended.
`monitor`	Same as `-m`.
`noclobber`	Same as `-C`.
`noexec`	Same as `-n`.
`noglob`	Same as `-f`.
`nolog`	Does not save function definitions in history file.
`notify`	Same as `-b`.
`nounset`	Same as `-u`.

`privileged` Same as `-p`.
`verbose` Same as `-v`.
`trackall` Same as `-h`.
`vi` Puts you in insert mode of a `vi`-style inline editor. Continues until you hit escape character (`033`). This puts you in move mode. A return sends the line.
`viraw` Processes each character as it is typed in `vi` mode.
`xtrace` Same as `-x`.

For `ksh -o`, the *option* argument is required. For `set -o`, if no option name is supplied, the current option settings are printed.

`-p` Disables processing of the `$HOME/.profile` file. Uses the file `/etc/suid_profile` instead of the `ENV` file. This mode is on whenever the effective uid (gid) is not equal to the real uid (gid). Turning this off causes the effective uid and gid to be set to the real uid and gid.

`-S` Prefixes commands with a date and time stamp of the form *day month date hh:mm:ss.*

`-s` Sorts the positional parameters lexicographically.
Note: For a decsription of `ksh -s`, see the Invocation subsection.

`-t` Exits after reading and executing one command.

`-u` Treats unset parameters as an error when substituting.

`-v` Prints shell input lines as they are read.

`-x` Prints commands and their arguments as they are executed.

`-` Turns off `-x`, `-v`, and `-S` flags and stops examining arguments for flags.

`--` Does not change any of the flags. This flag is useful in setting `$1` to a value beginning with `-`. If no arguments follow this flag, the positional parameters are unset.

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 `$-`. Unless `-A` is specified, the remaining arguments are positional parameters and are assigned, in order, to `$1 $2` . . . . If no arguments are specified, the names and values of all named parameters are printed on the standard output.

`setucat` *cat* Sets the active category. For usage and description, see `setucat`(1).

`setusrv` Sets the user's security attributes. For usage and description, see `setusrv`(1).

`setucmp` *cmp* Sets active compartments. Available only to the lowest-level login shell. For usage and description, see `setucmp`(1).

`setulvl` *level* Raises the security level. Available only to the lowest-level login shell. For usage and description, see `setulvl`(1).

† `shift` [*n*] The positional parameters from `$`*n*`+1` ... are renamed `$1` ...; default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a nonnegative number less than or equal to `$#`.

† `times` Prints the accumulated user and system times for the shell and for processes run from the shell.

† `trap` [*arg*] [*sig*] ...
arg is a command to be read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be specified as a number or as the name of the signal. 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. If *arg* is omitted or is `-`, all trap(s) *sig* 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 *sig* is `ERR`, *arg* will be executed whenever a command has a nonzero exit status. If *sig* is `DEBUG`, *arg* will be executed after each command. If *sig* is `0` or `EXIT` and the `trap` statement is executed inside the body of a function, the command *arg* is executed after the function completes. If *sig* is `0` or `EXIT` for a `trap` set outside any function, 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.

†† `typeset` [±`HLRZfilrtux`[*n*]] [*name*[=*value*]] ...
Sets attributes and values for shell parameters. When invoked inside a function, a new instance of the parameter *name* is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:

`-H` This flag provides UNIX system to host-name file mapping on non-UNIX system machines.

`-L` Left justifies and removes leading blanks from *value*. If *n* is nonzero, it defines the width of the field; otherwise, it is determined by the width of the value of first assignment. When the parameter is assigned to, it is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading 0's are removed if the `-Z` flag is also set. The `-R` flag is turned off.

`-R` Right justifies and fills with leading blanks. If *n* is nonzero, it defines the width of the field; otherwise, it is determined by the width of the value of first assignment. The field is left-filled with blanks or truncated from the end if the parameter is reassigned. The `L` flag is turned off.

`-Z` Right justifies and fills with leading 0's, if the first nonblank character is a digit and the `-L` flag has not been set. If *n* is nonzero, it defines the width of the field; otherwise, it is determined by the width of the value of first assignment.

`-f` The names refer to function names rather than parameter names. No assignments can be made and the only other valid flags are `-t`, `-u` and `-x`. The `-t` flag turns on execution tracing for this function. The `-u` flag causes this function to be marked undefined. The `FPATH` variable will be searched to find the function definition when the function is referenced. The `-x` flag allows the function definition to remain in effect across shell procedures invoked by name.

`-i` Specifies that parameter is an integer. This makes arithmetic faster. If *n* is nonzero, it defines the output arithmetic base; otherwise, the first assignment determines the output base.

`-l` Converts all uppercase characters to lowercase. The uppercase flag, `-u` is turned off.

`-r` The given *names* are marked read only; these names cannot be changed by subsequent assignment.

`-t` Tags the named parameters. Tags are user-definable and have no special meaning to the shell.

`-u` Converts all lowercase characters to uppercase characters. The lowercase flag, `-l`, is turned off.

`-x` Marks the given *name*s for automatic export to the environment of subsequently-executed commands.

Using `+` rather than `-` causes these flags to be turned off. If no *name* arguments are given but flags are specified, a list of names (and optionally the values) of the parameters that have these flags set is printed. (Using `+` rather than `-` keeps the values from being printed.) If no *name*s and flags are specified, the *names* and *attributes* of all parameters are printed.

`ulimit` [`-f`] [*n*] Imposes a size limit of *n* blocks. The `-f` option imposes a size limit of *n* 4096-byte blocks on files written by child processes (files of any size may be read). With no argument, the current limit in 4096-byte blocks is printed. See `ulimit`(2). If no option is specified, `-f` is assumed.

`umask` [*mask*] The user file-creation mask is set to *mask* (see `umask`(2)). *mask* can either be an octal number or a symbolic value as described in `chmod`(1). If a symbolic value is specified, the new umask value is the complement of the result of applying *mask* to the complement of the previous umask value. If *mask* is omitted, the current value of the mask is printed.

`unalias` *name* ...
The parameters given by the list of *name*s are removed from the *alias* list. See `unalias`(1).

`unset` [`-f`] *name* ...
The parameters given by the list of *name*s are unassigned; that is, their values and attributes are erased. Read-only variables cannot be unset. If the `-f` flag is set, the names refer to *function* names. Unsetting `ERRNO`, `LINENO`, `MAILCHECK`, `OPTARG`, `OPTIND`, `RANDOM`, `SECONDS`, `TMOUT`, and _ removes their special meaning, even if they are subsequently assigned to.

† `wait` [*job*] Waits for the specified *job* and reports its termination status. If *job* is not specified, all currently active child processes are awaited. The exit status from this command is that of the process awaited. For a format description of *job*, see the Jobs subsection. See `wait`(1).

`whence` [`-pv`] *name* ...
For each *name*, indicates how it would be interpreted if used as a command name.

The `-v` flag produces a more verbose report.

The `-p` flag does a path search for *name*, even if name is an alias, a function, or a reserved word.

NOTES

If this utility is installed with a privilege assignment list (PAL), a user with one of the following active categories is allowed to perform the actions shown:

Active Category	Action
`secadm`, `system`	Can redirect to or from any file using the facilities described in the Parameter Substitution, Quoting, and Input/Output subsections of this man page; change to any directory using the `cd`(1) command described in the File Name Generation subsection of this man page; kill any user process using the `kill`(1) command; arbitrarily set the shell process security attributes using the `setulvl`(1), `setucmp`(1), and `setusrv`(1) commands; and change process limits to any value using the `ulimit`(2) command.
`sysadm`	Constrained by security label restrictions but not by ownership, mode, and ACL considerations when redirecting to or from files using the facilities described in the Parameter Substitution, Quoting, and Input/Output sections of this man page; can change to directories using `cd`(1) command; can expand path names using the patterns described in the File Name Generation subsection of this man page; or can kill user processes using the `kill`(1) command. The `sysadm` administrator can change process limits to any value using the `ulimit`(2) command. However, this user can only set the shell process security attributes using `setulvl`(1), `setucmp`(1), or `setusrv`(1) in ways that are allowed to nonadministrative users.

`sysops` Constrained by security label restrictions but not by ownership considerations when killing processes using the `kill`(1) command and can change process limits to any value using the `ulimit`(2) command. The `sysops` administrator is treated as a nonadministrative user with respect all other shell activities.

If the `PRIV_SU` configuration option is enabled, the super user can override all `ksh` restrictions.

If a command that is a tracked alias 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 `-t` option of the `alias`(1) command to correct this situation.

Some very old shell scripts contain a `^` as a synonym for the pipe character (`|`).

Using the `fc` built-in command within a compound command will cause the entire command to disappear from the history file.

The built-in command `.` *file* reads the entire file before any commands are executed. Therefore, `alias` and `unalias` commands in the file will not apply to any functions defined in the file.

Traps are not processed while a job is waiting for a foreground process. Thus, a trap on `CHLD` will not be executed until the foreground job terminates.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. If a specified shell script could not be found by a noninteractive shell, the shell exits with 127. Otherwise, the shell returns the exit status of the last command executed (see also the `exit` command). If the shell is being used noninteractively, execution of the shell file is abandoned. Run-time errors detected by the shell are reported by printing the command or function name and the error condition. If the line number that the error occurred on is greater than one, the line number is also printed in square brackets (`[ ]`) after the command or function name.

EXAMPLES

Example 1: The following is an example of setting up `.profile`, which is the initialization file for `ksh`. It is executed once at login, and then executes the file to which `ENV` is set.

```
CDPATH=.:..:~:/         #path that cd uses
FCEDIT=vi               #command line editor
ENV=~/.env              #environment file
HISTSIZE=32             #ksh saves last 32 commands
MAILCHECK=300           #ksh checks for mail every 5 minutes
set -o ignoreeof        #ignore ^D on login shell
TERM='tset - -Q '?ampex230''
stty erase '^?' kill '^u' intr '^c' echo

export CDPATH EDITOR FCEDIT CRAY ENV HISTSIZE MAILCHECK TERM
```

Example 2: The following is an example of a `.env` file that is executed each time a new `ksh` is run; it allows all aliases to be carried to all new shells.

```
#
alias -x lsf='/bin/ls -CF'
alias -x lsl='/bin/ls -lgF'
alias -x h='fc -lr'
alias -x hf='fc -l $HISTSIZE | more'
alias -x pe=printenv
```

FILES

`/etc/profile`	File containing system default shell startup commands
`$HOME/.profile`	File containing user's shell startup commands
`$TMPDIR/sh*`	Temporary working files
`/dev/null`	Zero-length file
`a.out`	Executable binary file

SEE ALSO

`alias`(1), `bg`(1), `cd`(1), `chown`(1), `command`(1), `echo`(1), `emacs`(1), `env`(1), `fc`(1), `fg`(1), `getopts`(1), `jobs`(1), `kill`(1), `login`(1), `pwd`(1), `read`(1), `setucat`(1), `setucmp`(1), `setusrv`(1), `test`(1), `unalias`(1), `vi`(1), `wait`(1)

`dup`(2), `exec`(2), `execve`(2), `fork`(2), `pipe`(2), `setuid`(2), `signal`(2), `umask`(2), `ulimit`(2), `wait`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

`a.out`(5), `profile`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

`dmmode`(1) Online only

`environ`(7) Online only

Learning the Korn Shell, Bill Rosenblatt, O'Reilly & Associates, Inc., 1990

The KornShell Command and Programming Language, Morris Bolsky and David Korn, Prentice Hall, 1989

General UNICOS System Administration, Cray Research publication SG–2301

NAME

ksrvtgt – Uses a service key to fetch and store a Kerberos ticket-granting ticket

SYNOPSIS

ksrvtgt*name instance* [[*realm*] *srvtab*]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The ksrvtgt utility retrieves a ticket-granting ticket with a lifetime of 5 minutes for the principal *name.instance@realm*. It uses the service key found in the *srvtab* file to decrypt the response and stores the ticket in the standard ticket cache.

The ksrvtgt utility accepts the following options:

name instance
Specifies the principal for which the ticket-granting ticket is retrieved.

realm Specifies the realm of the principal for which the ticket-granting ticket is retrieved. If you omit *realm* on the command line, the local realm is used.

srvtab Specifies the file that contains the service key that decrypts the response. If you omit *srvtab* on the command line, the /etc/srvtab file is used.

The ksrvtgt utility is primarily for use in shell scripts and other batch-type facilities.

MESSAGES

The Generic kerberos failure (kfailure) message can indicate a whole range of problems, the most common of which is the inability to read the service key file.

FILES

/etc/krb.conf	File that gets the name of the local realm
/etc/srvtab	Default service key file
/tmp/tkt[*uid*]	Default ticket file

SEE ALSO

kdestroy(1), kinit(1)

kerberos(7) (available only online)

NAME

ksu – Uses Kerberos to substitute user ID

SYNOPSIS

ksu [-] [*name* [*args*]]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The ksu utility allows you to become another user without logging off. The default user *name* is root.

To use ksu, the appropriate password must be supplied (unless you are an appropriately authorized user). If the password is correct, ksu executes a new shell with the real and effective user ID set to that of the specified user. The new shell is the optional program named in the shell field of the specified user's password file entry (see udb(5)), or /bin/sh if none is specified (see sh(1)). To restore your original user identity, exit the new shell.

Any additional arguments specified on the command line are passed to the program invoked as the shell. For example, when sh(1) is used, an argument of the form -c *string* executes *string* via the shell and an option of -r gives the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like sh(1). If the first argument to ksu is a -, the environment is changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is -, causing 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. If the optional program used as the shell is /bin/sh, the user's .profile can check *arg0* for -sh or -ksu to determine whether it was invoked by login(1) or ksu, respectively. If the user's program is not /bin/sh, the program is invoked with an *arg0* of -*program* by both login and ksu.

The ksu utility accepts the following options:

- Changes environment to that of specified user name.

name Indicates user name to log on to (default is root).

args Specifies shell arguments for new login.

Only users with root instances listed in the `.klogin` file can use the `ksu` utility to change to `root` (the `klogin`(1) utility describes the format of this file). When you attempt root access, `ksu` attempts to fetch a ticket-granting ticket for *username.root@localrealm*; *username* is the user name of the process. If possible, the tickets are used to obtain, use, and verify tickets for the `rcmd.host@`*localrealm* service; `host` is the canonical host name of the machine (which is the first field, lower case, of the domain name). If this verification fails, the `ksu` utility is disallowed. If the `rcmd.host@`*localrealm* service is not registered, the `ksu` utility is allowed.

By default (unless the prompt is reset by a startup file), the super-user prompt is set to #.

When not attempting to switch to the `root` user, `ksu` behaves exactly like `su`(1).

SEE ALSO

`csh`(1), `klogin`(1), `login`(1), `passwd`(1), `sh`(1), `su`(1)

`group`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

`environ`(7) (available only online)

NAME

`last` – Indicates the last logins of users and teletypes

SYNOPSIS

`/usr/ucb/last` [`-`*number*] [`-f` *file*] [*names*] [*ttys*]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `last` utility looks in the `/etc/wtmp` file for information about a user, a teletype, or any group of users and teletypes. The `wtmp` file records all logins and logouts that have occurred since the last initialization of the file. Arguments specify names of users or teletypes of interest. Names of teletypes can be specified fully or abbreviated. For example, `last 0` is the same as `last tty0`. If you specify multiple arguments, `last` prints the information applying to any of the arguments. The `last` utility prints the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype on which the session occurred. `last` indicates whether the session is still continuing or was cut short by a reboot.

The pseudo-user `reboot` logs in when the system is rebooted. Thus, the following command line indicates the mean time between reboots:

```
last reboot
```

The `last` utility accepts the following options:

`-`*number* Limits the number of entries displayed to *number*.

`-f` *file* Uses *file* as the name of the accounting file instead of `/etc/wtmp`. *file* must be in `wtmp`(5) format.

names Logins to be checked.

ttys Teletypes to be checked.

EXAMPLES

Example 1: To list all of the super user's sessions, as well as all sessions on the console terminal, enter the following:

```
last root console
```

Example 2: To print a record of all logins and logouts in reverse order, enter the following without arguments:

```
last
```

FILES

`/etc/wtmp` Login database

SEE ALSO

`utmp`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

ld – Invokes the link editor

SYNOPSIS

ld [-D *dirstring*] [-e *name*] [-F] [-g] [-i] [-j *names*] [-l *names*] [-L *ldirs*] [-m] [-n] [-o *outfile*] [-r] [-s] [-u *unames*] [-V] [-Z] [-z *file*] *files*

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The ld command links relocatable object modules to produce an executable program. This command invokes the same loader as does segldr(1), but with a traditional UNIX ld invocation.

The *files* specified on the command line may be either sequential object files created by the compilers or assembler, object libraries created by bld(1) or files containing loader directives. Files ending with .o will be treated as bin files. Files ending with .a will be treated as lib files. You can intersperse file names with options on the command line.

The ld command accepts the following options:

-D *dirstring* Specifies a character string composed of loader directives separated by semicolons. The loader processes directives supplied with -D before it processes any directives files.

-e *name* Sets the program entry address to the value of the symbol *name*.

-F Enables default library processing. The standard system libraries are processed after any user-supplied libraries. Processing of the system libraries is disabled by default.

-g Generates the Debug Symbol tables and appends them to the executable program. This option is enabled by default. See the -s option.

-i Generates a shared-text program on Cray PVP systems. This option is equivalent to the -n option.

-j *names* List of directives file names, separated by commas. When a name begins with a dot (.) or a slash (/), ld assumes it is a complete path name and uses it without modification. Otherwise, ld checks for a segdirs/*name* file in the list of search directories and uses the first one found. See the -L option for the list of search directories.

-l *names* Identifies library files. When a name begins with a dot (.) or a slash (/), it is assumed be a complete path name and is used without modification. Otherwise, the ld command checks first for files /opt/ctl/craylibs/craylibs/lib*name*.a and /lib/lib*name*.a, and then for file /usr/lib/lib*name*.a. It uses the first file it finds. See the -L option.

`-L` *ldirs* Changes the `-l` option search algorithm to look for library files in directories *ldirs* before looking in the `/opt/ctl/craylibs/craylibs`, `/lib`, or `/usr/lib` directories. If the `-F` option is used to include the system default directories, the loader searches directories *ldirs* for those libraries before searching the `/opt/ctl/craylibs/craylibs`, `/lib`, or `/usr/lib` directories. Multiple `-L` options are cumulative.

`-m` Generates a load map of the executable program and writes it to the `stdout` file.

`-n` Generates a shared-text program on Cray PVP systems. This option is equivalent to the `-i` option.

`-o` *outfile* Writes the executable program to *outfile*. The default *outfile* name is `a.out`.

`-r` Produces a relocatable output from `.o` files. That is, instead of generating an executable, it generates one relocatable combining the .o files named. The output is suitable for use by another invocation of `ld`. Equivalent to using the following directives:

```
OUTFORM=REL
USX=NOTE
SYSTEM=STDALONE
ZSYMS=OFF
```

`-s` Inhibits the generation of the Debug Symbol tables.

`-u` *unames* Enters *unames* as undefined symbols. This is useful for loading from a library, because undefined symbols are needed to force the loading of desired routines.

`-V` Lists the SEGLDR version line on the `stderr` file.

`-Z` Inhibits the loader from reading the default directives file. The default directives file is either `/opt/ctl/craylibs/craylibs/segdirs/opt_defld` or `/lib/segdirs/def_ld`. The default directives file is required for configuring programs correctly for execution under the UNICOS operating system. The `-Z` option should be used only by special-purpose programs.

`-z` *file* Specifies an alternative default directives file. The alternative directives must configure the program correctly for execution under the UNICOS operating system.

files Specifies the files to be loaded.

ENVIRONMENT VARIABLES

The `ld` command looks for and processes the following environment variables:

`LDDIR` Contains one or more strings separated by semicolons. Each string may be either a `ld` directive or the name of a file containing `ld` directives.

`TMPDIR` Specifies the directory that the loader uses for its temporary file.

`LPP` Specifies the number of lines to print on each page of listing output. The value must be between 15 and 999, and the default is 57.

MSG_FORMAT Describes a format specification similar to that of the C library routine `printf`; this specification can be used to alter `ld` error message displays.

NLSPATH Specifies a list of alternate directories that the loader should search for its error message catalog. It is used to select alternate catalogs for debugging, or when different versions of `ld` are operating on the same system. `NLSPATH` is not needed for normal operations.

TARGET Specifies the machine characteristics of the system on which the program will execute. If the `TARGET` variable has not been specified, the program will be adapted to the host system.

The following defaults for loader directives are automatically used when you invoke `ld`:

```
force=on
duporder=on
nodeflib
dupentry=caution:note:note
usx=warning
```

MESSAGES

The full range of error messages and the proper responses are listed in the *Segment Loader (SEGLDR) and ld Reference Manual*, Cray Research publication SR–0066.

FILES

`a.out`	Executable program
file`.o`	Relocatable object file
`/opt/ctl/craylibs/craylibs/libf.a`	Fortran library
`/opt/ctl/craylibs/craylibs/libfi.a`	Fortran intrinsic library
`/opt/ctl/craylibs/craylibs/libm.a`	Math library
`/opt/ctl/craylibs/craylibs/libsci.a`	Scientific library
`/lib/libc.a`	C library
`/opt/ctl/CC/CC/lib/libC.a`	C++ library (only if your site has a C++ license)
`/lib/libp.a`	Pascal library
`/opt/ctl/craylibs/craylibs/libu.a`	Utility library
`/lib/segdirs/def_ld` and `/opt/ctl/craylibs/craylibs/segdirs/opt_defld`	Default directives files

SEE ALSO

`ar`(1) archive and library maintainer for portable archives
`bld`(1) maintains relocatable libraries
`nm`(1) prints name list from load modules
`segldr`(1) invokes the Cray Research segment loader (SEGLDR)

`cc`(1) invokes the Cray Standard C compiler and is described in the *Cray Standard C Reference Manual*, Cray Research publication SR–2074

`f90`(1) invokes the CF90 compiler and is described in the *CF90 Commands and Directives Reference Manual*, Cray Research publication SR–3901

`a.out`(5) describes the loader output file
`relo`(5) describes the relocatable object table format under the UNICOS operating system
`taskcom`(5) describes the task common table format
in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

Segment Loader (SEGLDR) and ld Reference Manual, Cray Research publication SR–0066

NAME

`lex` – Generates programs for simple lexical tasks

SYNOPSIS

```
lex [-t] [-r] [-n] [files]
lex [-t] [-r] [-v] [files]
```

Obsolescent version; may not be supported in future releases:

```
lex -c [-t] [-r] [-n] [files]
lex -c [-t] [-r] [-v] [files]
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4
AT&T extensions (`-r` option)

DESCRIPTION

The `lex` utility generates programs to be used in simple lexical analysis of text.

The input files (standard input by default) contain strings and expressions for which to search and C text to be executed when strings are found.

The `lex` utility accepts the following options and operand:

`-t` Causes the `lex.yy.c` program to be written to standard output.

`-r` Indicates RATFOR actions (RATFOR, a Rational Fortran compiler, is not supported by Cray Research.)

`-n` Does not print the `-v` summary.

`-v` Provides a one-line summary of statistics of the machine generated. Multiple files are treated as a single file. When files are not specified, standard input is used.

`-c` (Obsolescent) Indicates C-language actions and is the default.

files A path name of one or more input files. If more than one such file is specified, all files are concatenated to produce a single `lex` program. If no *files* are specified, or if the operand is `-`, the standard input is used.

A file, `lex.yy.c`, is generated; when loaded with the library, it 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 brackets to indicate character classes, as in `[abx-z]` to indicate `a`, `b`, `x`, `y`, and `z`. The `*`, `+`, and `?` operators mean, respectively, any nonnegative 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 newline characters.

Parentheses for grouping and the vertical bar for alternation are also supported. The notation *r* {`d`, `e`} in a rule indicates between `d` and `e` instances of extended regular expression *r* . It has higher precedence than `|`, but lower than `(`, `**`, `?` , `+`, and concatenation. Thus, `[a-zA-Z]+` matches a string of letters. The `^` character at the beginning of an expression permits a successful match only immediately after a newline character, and `$` at the end of an expression requires a trailing newline character.

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 when it is enclosed by `"` symbols or preceded by `\`.

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()` routine that calls it. The `REJECT` action on the right side of the rule causes this match to be rejected and the next suitable match executed; the `yymore()` function accumulates additional characters into the same `yytext` array; and the `yyless(p)` function pushes back the portion of the string matched beginning at *p*, which should be between `yytext` and `yytext` + `yyleng`. The `input` and `output` macros, defaulted to `stdin` and `stdout`, respectively, use files `yyin` and `yyout` to read from and write to, respectively.

Any line beginning with a `<blank>` is assumed to contain only C text and is copied; when it precedes `%%`, it is copied into the external definition area of the `lex.yy.c` file. All rules should follow a `%%` (as is done in `yacc`(1)). Lines that precede `%%` and begin other than a `<blank>` character define the string on the left as the remainder of the line; the string can be called out later if it is surrounded with `{}`. Braces do not imply parentheses; only string substitution is done.

Certain table sizes for the resulting finite-state machine can be set in the definitions section as follows:

`%p` *n* Number of positions is *n* (default is 2000).

`%n` *n* Number of states is *n* (default is 500).

`%t` *n* Number of parse tree nodes is *n* (default is 1000).

`%a` *n* Number of transitions is *n* (default is 3000).

The use of one or more of these automatically implies the `-v` option, unless the `-n` option is used.

NOTES

The external names generated by `lex` all begin with the prefix `yy` or `YY`.

EXIT STATUS

The `lex` utility exits with one of the following values:

0 Successful completion.

>0 An error occurred.

BUGS

Extended Regular Expressions (EREs) are currently not supported in `lex`.

EXAMPLES

Example 1: The following example is an input program for `lex`:

```
D    [0-9]
%%
if      printf("IF statement0);
[a-z]+  printf("tag, value %s0,yytext);
0{D}+   printf("octal number %s0,yytext);
{D}+    printf("decimal number %s0,yytext);
"++"    printf("unary op0);
"+"     printf("binary op0);
"/*" skipcommnts();
%%
  skipcommnts()
 {
        for (;;)
        {
                while (input() != '*')
                        ;
         if (input() != '/')
                        unput(yytext[yyleng-1]);
                else
                        return;
         }
 }
```

SEE ALSO

yacc(1)

lex & yacc, Doug Brown and Tony Mason, O'Reilly & Associates, Inc., 1992.

The UNIX Programming Environment, Brian W. Kernighan and Rob Pike, Prentice-Hall, Inc., 1984.

NAME

`limit` – Sets resource limits

SYNOPSIS

`limit` [`-j` *jid*] [`-c` *cputimelim*] [`-m` *memorylim*] [`-e` *mpppelim*] [`-t` *mpptimelim*] [`-s` *socketbuflim*] [`-v`]

`limit -p` *pid* [`-c` *cputimelim*] [`-m` *memorylim*] [`-d` *corelim*] [`-f` *openfilelim*] [`-t` *mpptimelim*] [`-s` *socketbuflim*] [`-v`]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `limit` utility establishes limits on resource usage for a process or a job.

The `limit` utility accepts the following options:

`-c` *cputimelim* Indicates a limit on CPU time. The *cputimelim* argument refers to CPU seconds. A *cputimelim* of 0 indicates an unlimited amount of CPU time. If the `-c` option is not specified, the CPU time limit is not modified. (See the CAUTIONS section.)

`-d` *corelim* Indicates a limit on core file sizes. The *corelim* argument refers to memory words and is rounded up to the nearest click boundary. There are 512 decimal words per click on Cray Research systems. A *corelim* of 0 indicates the maximum core file size allowed. Specifying a *corelim* value less than the size of the process will result in a truncated core file that consists only of the user common structure and the user area. Specifying a *corelim* of `nocore` will inhibit the creation of a core file altogether. This option is supported only for processes.

`-e` *mpppelim* Indicates a limit on the number of PEs. This option may be specified only with the `-j` option.

`-f` *openfilelim* Indicates a limit on the maximum number of files that a process can have open at any given time. This limit is supported only with the `-p` option. The limit may be from 64 through the user's defined limit in the user database (UDB); the default is 64. This limit is applied only to the children of the process specified.

`-j` *jid* Indicates a particular job. A *jid* of 0 means the current job. The `-j` option may not be specified with the `-p` option. If neither the `-p` nor the `-j` option is specified, a default of `-j 0` is used.

`-m` *memorylim* Indicates a limit on memory size. The *memorylim* argument refers to memory words. The given *memorylim* is rounded up to the nearest click boundary. There are 512 decimal words per click on Cray Research systems. A *memorylim* of 0 indicates the maximum memory size available. If the `-m` option is not specified, the memory size limit is not modified.

`-p` *pid* Indicates a particular process. A *pid* of 0 means the current process. The `-p` option may not be specified with the `-j` option.

`-s` *socketbuflim* Indicates a limit on per session socket buffer (sockbuf) space. The per session sockbuf space is the sum of the sockbuf space requested by all of the sockets used by the session. The *socketbuflim* argument refers to memory clicks. There are 512 decimal words per click on Cray Research systems. A *socketbuflim* of 0 indicates no space limit. If the `-s` option is not specified, the sockbuf space limit is not modified.

`-t` *mpptimelim* Indicates a limit on processing element (PE) time.

`-v` Writes the previous time, memory size, and core file size limits to standard output in a more verbose mode.

Any user may change a limit to be more restrictive. Only an appropriately authorized user can increase resource limits. Only an appropriately authorized user can set the resource limits of another user, process, or session. Limits are inherited by child processes.

The *pid*, *jid*, *cputimelim*, *corelim*, *memorylim*, and *socketbuflim* arguments have the following characteristics in common. All must be nonnegative integer values. If the argument contains a leading `0x` or `0X`, it is evaluated as hexadecimal. If the argument contains a leading zero, it is evaluated as octal. Otherwise, it is evaluated as decimal.

NOTES

If this utility is installed with a privilege assignment list (PAL), a user with one of the following active categories is allowed to perform the actions shown:

Active Category	Action
`system`, `secadm`	Allowed to raise or lower the resource limits of any user, process, or session.
`sysadm`	Allowed to raise or lower the resource limits of any user, process, or session, subject to security label restrictions. Shell-redirected I/O is subject to security label restrictions.

If the `PRIV_SU` configuration option is enabled, the super user is allowed to raise or lower the resource limits of any user, process, or session.

If a process exceeds the process CPU time limit, `SIGCPULIM` is sent to the offending process. By default, the `SIGCPULIM` will terminate the process and the parent shell will recognize the `SIGCPULIM` and send an error message to standard error. If the job CPU time limit is exceeded, `SIGCPULIM` is sent to all processes in the job, which includes the parent shell. In this case, no error message will be sent to standard error.

CAUTIONS

If more than one call is made to `limit` at nearly the same time to modify the same entity, there is potential for unpredictable results. The CPU time limit does not apply when running as root.

EXIT STATUS

If `limit` succeeds, several decimal integers (separated by a newline character) are written to `stdout`. If core files are disabled, the string `nocore` is printed.

When the job mode (`-j`) option is specified, the following values are written to `stdout`:

- CPU seconds
- Words of memory
- Socket buffer limit
- Number of PE limit
- PE time limit

When the process mode (`-p`) option is specified, the following values are written to `stdout`:

- CPU seconds
- Words of memory
- Core file limit
- File descriptor limit
- Socket buffer limit
- PE time limit

The `limit` utility returns an exit status of 0 upon success.

If `limit` fails, an appropriate error message is written to `stderr`, and a nonzero exit status is returned. When `limit` fails, none of the limits are modified, if possible.

SEE ALSO

`nlimit`(1)

`limit`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

`line` – Reads one line

SYNOPSIS

```
line
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

XPG4

DESCRIPTION

The `line` utility copies one line (up to a `<newline>` character) from the standard input and writes it on standard output.

It returns an exit code of 1 on `EOF` and always prints at least a `<newline>` character. It is often used in shell scripts to read from the user's terminal.

NOTES

If this utility is installed with a privilege assignment list (PAL), a user with one of the following active categories is allowed to perform the actions shown:

Active Category	Action
`system`, `secadm`	In a privileged administrator shell environment, shell-redirected I/O is not subject to file protections.

If the `PRIV_SU` configuration option is enabled, shell-redirected I/O on behalf of the super user is not subject to file protections.

The `read`(1) utility is the preferred method of obtaining input from a user's terminal.

EXIT STATUS

The `line` utility exits with one of the following values:

0 Successful completion.

>0 An error occurred.

EXAMPLES

The following standard shell (sh(1)) script reads input lines from stdin and writes them to file ofile:

```
echo "Enter text.  End with CONTROL-d"
while REC="`line`"
do
        echo "${REC}" >> ofile
done
```

SEE ALSO

read(1), sh(1)

read(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

`lint` – A C-language program checker

SYNOPSIS

`lint` [`-a`] [`-b`] [`-c`] [`-D` *name* = *value*] [`-F`] [`-h`] [`-I` *directory*] [`-k`] [`-L` *directory*] [`-m`] [`-n`] [`-o` *x*] [`-p`] [`-s`] [`-u`] [`-U` *name*] [`-x`] [`-v`] [`-V`] [`-y`] *files*

IMPLEMENTATION

All Cray Research systems

STANDARDS

XPG4 AT&T extensions (`-F`, `-h`, `-k`, `-m`, `-s`, `-V`, and `-y` options)

DESCRIPTION

`lint` detects features of C program files that are likely to be bugs, nonportable, or wasteful. It also checks type usage more strictly than the compiler. `lint` issues error and warning messages. Among the things it detects are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. `lint` checks for 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) utility when given a `.c` file as input. Files with other suffixes are warned about and ignored.

`lint` takes all the `.c`, `.ln`, and `llib-l`*x*`.ln` (specified by `-l`*x*) files and processes them in their command-line order. By default, `lint` appends the standard C `lint` library (`llib-lc.ln`) to the end of the list of files. When the `-c` option is used, the `.ln` and the `llib-l`*x*`.ln` files are ignored. When the `-c` option is not used, the second pass of `lint` checks the `.ln` *and the* `llib-l`*x*`.ln` list of files for mutual compatibility.

Any number of `lint` options may be used, in any order, intermixed with file name arguments. The following options are supported:

`-a` Suppresses complaints about assignments of long values to variables that are not long.

`-b` Suppresses complaints about `break` statements that cannot be reached.

`-h` Does not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

`-m` Suppresses complaints about external symbols that could be declared static.

`-u` Suppresses 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` Does not report variables referred to by external declarations but never used.

The following arguments alter `lint`'s behavior:

`-I` *directory*
Searches for included header files in the directory *directory* before searching the current directory and/or the standard place.

`-l`*x* Includes the `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.

`-L` *directory*
Searches for `lint` libraries in *directory* before searching the standard place.

`-n` Does not check compatibility against the standard C `lint` library.

`-p` Attempts to check portability to other dialects of C. Along with stricter checking, this option causes all nonexternal names to be truncated to 8 characters and all external names to be truncated to 6 characters and one case.

`-s` Produces one-line diagnostics only. `lint` occasionally buffers messages to produce a compound report.

`-k` Alters the behavior of `/*LINTED [`*message*`]*/` directives. Normally, `lint` will suppress warning messages for the code following these directives. Instead of suppressing the messages, `lint` prints an additional message containing the comment inside the directive.

`-y` Specifies that the file being checked by `lint` will be treated as if the `/*LINTLIBRARY*/` directive had been used. A `lint` library is normally created by using the `/*LINTLIBRARY*/` directive.

`-F` Prints path names of files. `lint` normally prints the file name without the path.

`-c` Causes `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 interfunction compatibility.

`-o` *x* Causes `lint` to create a `lint` library with the name `llib-l`*x*`.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`*x*`.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.

Some of the above settings are also available through the use of "`lint` comments" (see below).

`-V` Writes to standard error the product name and release.

The following options are not for general use:

`-R` *file* Writes a `.ln` file to *file*, for use by `cxref`(1).

`-W` *file* Writes a `.ln` file to *file*, for use by `cflow`(1).

`lint` recognizes many `cc`(1) command-line options, including `-D`, `-U`, `-g`, and `-O`, although `-g` and `-O` are ignored. Unrecognized options are warned about and ignored. The predefined macro `lint` is defined to allow certain questionable code to be altered or removed for `lint`. Thus, 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`:

`/*ARGSUSED`*n*`*/`	Makes `lint` check only the first `n` arguments for usage; a missing *n* is taken to be 0 (this option acts like the `-v` option for the next function).
`/*CONSTCOND*/` or `/*CONSTANTCOND*/` or `/*CONSTANTCONDITION*/`	Suppresses complaints about constant operands for the next expression.
`/*EMPTY*/`	Suppresses complaints about a null statement consequent on an if statement. This directive should be placed after the test expression, and before the semicolon. This directive is supplied to support empty if statements when a valid else statement follows. It suppresses messages on an empty `else` consequent.
`/*FALLTHRU*/` or `/*FALLTHROUGH*/`	Suppresses complaints about fall through to a `case` or `default` labeled statement. This directive should be placed immediately preceding the label.
`/*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.
`/*LINTED` [*message*]`*/`	Suppresses any intrafile warning except those dealing with unused variables or functions. This directive should be placed on the line immediately preceding where the `lint` warning occurred. The `-k` option alters the way in which `lint` handles this directive. Instead of suppressing messages, `lint` will print an additional message, if any, contained in the comment. This directive is useful in conjunction with the `-s` option for post-`lint` filtering.
`/*NOTREACHED*/`	At appropriate points, stops comments about unreachable code. (This comment is typically placed just after calls to functions like `exit`(2)).
`/*PRINTFLIKE`*n*`*/`	Makes `lint` check the first (*n*–1) arguments as usual. The *n*th argument is interpreted as a `printf` format string that is used to check the remaining arguments.

/*PROTOLIB*n*/	Causes lint to treat function declaration prototypes as function definitions if *n* is nonzero. This directive can only be used in conjunction with the LINTLIBRARY directive. If *n* is zero, function prototypes will be treated normally.
/*SCANFLIKE*n*/	Makes lint check the first (*n*–1) arguments as usual. The *n*th argument is interpreted as a scanf format string that is used to check the remaining arguments.
/*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. The use of the ellipsis terminator (...) in the definition is suggested in new or updated code.

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, if -s is not specified. 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 that 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 interfile inconsistencies. This scheme works well with make(1); it allows make(1) to be used to lint only the source files that have been modified since the last time the set of source files were checked by lint.

FILES

LIBDIR	The directory where the lint libraries specified by the -l*x* option must exist (default is /usr/lib/lint)
LIBDIR/lint[12]	First and second passes
LIBDIR/llib-lc.ln	Declarations for C library functions (binary format; source is in LIBDIR/llib-lc)
LIBPATH/llib-lm.ln	Declarations for math library functions (binary format; source is in LIBDIR/llib-lm)
TMPDIR/**lint*	Temporary files

SEE ALSO

cc(1), cflow(1), cxref(1), make(1)

exit(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

lmcksum – Checksums the FLEXlm license file

SYNOPSIS

lmcksum [-c *license_file*] [-k]

IMPLEMENTATION

All supported platforms

DESCRIPTION

The lmcksum command computes the checksum of the portions of the flexible license manager (FLEXlm) license file that end users cannot change. lmcksum computes a checksum for each line in the license file and an overall checksum of the license file. The following fields are used when creating the checksum:

- *hostid* on the server lines
- *daemon_name* on the daemon lines
- *feature_name*, *version*, *daemon_name*, *expiration_date*, *number_of_licenses*, *encryption_code*, *vendor_string*, and *hostid* on feature lines

The lmcksum command is available only in the FLEXlm v2.4 release and later. lmcksum is either a link to or a copy of the lmutil(1) utility.

The lmcksum command accepts the following options:

-c *license_file*
: Uses the specified *license_file* as input. If you omit the -c option, lmcksum uses the LM_LICENSE_FILE environment variable to find the license file to use. If that environment variable is not set, lmcksum uses the /usr/local/flexlm/licenses/license.dat file. If you omit the *license_file* argument, lmcksum uses the LM_LICENSE_FILE environment variable. If that environment variable is not set, lmcksum uses the /usr/local/flexlm/licenses/license.dat file.

-k
: Case-sensitive checksum; computes the checksum by using the exact case of the feature line(s) encryption codes.

SEE ALSO

lmutil(1) for information about the core FLEXlm utility

license.dat(5) for information about the license configuration file for FLEXlm licensed applications in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

lmdown – Shuts down all FLEXlm license daemons gracefully

SYNOPSIS

lmdown [-c *license_file*] [-q]

IMPLEMENTATION

All supported platforms

DESCRIPTION

The lmdown command sends a message to all flexible license manager (FLEXlm) license daemons asking them to shut down. The license daemons write out their last messages to the log file, close the file, and exit. All licenses that the license daemons have provided are rescinded, so that the next time a client program verifies the license, it will not be valid. In the FLEXlm v2.4 release and later, lmdown is either a link to or a copy of the lmutil(1) utility.

The lmdown command accepts the following options:

-c *license_file*
: Uses the specified *license_file* as input. On UNICOS systems, to avoid affecting other license managers, always use the -c option or the LM_LICENSE_FILE environment variable. If you omit the -c option, lmdown uses the LM_LICENSE_FILE environment variable to find the license file to use. If that environment variable is not set, lmdown uses the /usr/local/flexlm/licenses/license.dat file. If you omit the *license_file* argument, lmdown uses the LM_LICENSE_FILE environment variable. If that environment variable is not set, lmdown uses the /usr/local/flexlm/licenses/license.dat file.

-q
: (Quiet mode) If you specify this option, lmdown will not ask for confirmation before asking the license daemons to shut down. If you omit this option, lmdown asks for confirmation before asking the license daemons to shut down.

SEE ALSO

lmgrd(1) for information about invoking the FLEXlm daemon
lmreread(1) for information about instructing the FLEXlm license daemon to reread the license file
lmstat(1) for information about reporting status of FLEXlm license daemons and feature usage
lmutil(1) for information about the core FLEXlm utility

license.dat(5) for information about the license configuration file for FLEXlm licensed applications in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

lmgrd – Invokes the FLEXlm license daemon

SYNOPSIS

lmgrd [-2] [-b] [-c *license_file*] [-d] [-l *logfile*] [-p] [-s *interval*] [-t *timeout*]

IMPLEMENTATION

All supported platforms

DESCRIPTION

The lmgrd command invokes the flexible license manager (FLEXlm) license daemon and is the main daemon program for the FLEXlm distributed license management system. When you invoke lmgrd, it looks for a license file that contains all required information about vendors and features.

The lmgrd command accepts the following options:

-2 Specifies startup arguments; if you use the -p option, the -2 option is required. The -2 option is the opposite of the -b option. Available in lmgrd v2.4 and later.

-b Specifies backward-compatibility mode. In FLEXlm v2.4 or later, the -b option is the default. If you are running a v2.1 or later version of lmgrd with a v1.5 or earlier vendor daemon, use this option.

-c *license_file* Uses the specified *license_file* as input. If you omit the -c option, lmgrd uses the LM_LICENSE_FILE environment variable to find the license file to use. If that environment variable is not set, lmgrd uses the /usr/local/flexlm/licenses/license.dat file. If you omit the *license_file* argument, lmgrd uses the LM_LICENSE_FILE environment variable. If that environment variable is not set, lmgrd uses the /usr/local/flexlm/licenses/license.dat file.

-d Specifies that the host names that are read from the license file should have the local domain name appended to them before sending to a client. This option is useful when clients are accessing licenses from another domain. Available in lmgrd v2.4 and later.

-l *logfile* Specifies the output log file to use; the output log file is sent to stdout by default.

-p Specifies that the lmdown(1) and lmremove(1) commands can be run only by a license administrator; if you use the -p option, the -2 option is required. A *license administrator* is a member of the lmadmin group, or, if the lmadmin group does not exist, a member of group 0. Available in lmgrd v2.4 and later.

-s *interval* Specifies the log file time-stamp interval (in minutes). The default value is 360 minutes.

`-t` *timeout* Specifies the time-out interval (in seconds) during which daemons must complete their connections to each other. The default value is 10 seconds. If the daemons are being run on busy systems or a very heavily loaded network, you may want to use a larger value.

SEE ALSO

`lmdown`(1) for information about shutting down all FLEXlm license daemons gracefully
`lmstat`(1) for information about reporting status of FLEXlm license daemons and feature usage

`license.dat`(5) for information about the license configuration file for FLEXlm licensed applications in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

`lmhostid` – Displays the host ID of a system

SYNOPSIS

```
lmhostid
```

IMPLEMENTATION

All supported platforms

DESCRIPTION

The `lmhostid` command calls the flexible license manager (FLEXlm) version of `gethostid`(2) and displays the results. In the FLEXlm v2.4 release and later, `lmhostid` is either a link to or a copy of the `lmutil`(1) utility.

The output of `lmhostid` looks like the following display:

```
lmhostid - Copyright (C) 1989-1994 Globetrotter Software, Inc.
The FLEXlm host ID of this machine is "3e9"
```

SEE ALSO

`lmutil`(1) for information about the core FLEXlm utility

NAME

lmremove – Removes specific FLEXlm licenses and returns them to license pool

SYNOPSIS

lmremove [-c *license_file*] *feature user host_name display*

IMPLEMENTATION

All supported platforms

DESCRIPTION

The lmremove command lets the system administrator remove a single user's flexible license manager (FLEXlm) license for a specified feature. This removal might be required if the licensed user was running the software on a node that subsequently crashed. This situation sometimes causes the license to remain unusable. lmremove allows the license to return to the pool of available licenses. In the FLEXlm v2.4 release and later, lmremove is either a link to or a copy of the lmutil(1) utility.

The lmremove command accepts the following arguments:

-c *license_file* Uses the specified *license_file* as input. If you omit the -c option, lmremove uses the LM_LICENSE_FILE environment variable to find the license file to use. If that environment variable is not set, lmremove uses the /usr/local/flexlm/licenses/license.dat file. If you omit the *license_file* argument, lmremove uses the LM_LICENSE_FILE environment variable. If that environment variable is not set, lmremove uses the /usr/local/flexlm/licenses/license.dat file.

feature Specifies the feature name from the FLEXlm license file. You must enter it exactly as the lmstat(1) command displays it.

user Specifies the user name of the license to be removed. You must enter it exactly as the lmstat(1) command displays it.

host_name Specifies the name of the system user from which the user is using the license. You must enter it exactly as the lmstat(1) command displays it.

display Specifies the name of the user's X Windows System display. You must enter it exactly as the lmstat(1) command displays it.

SEE ALSO

lmstat(1) for information about reporting status of FLEXlm license daemons and feature usage
lmutil(1) for information about the core FLEXlm utility

license.dat(5) for information about the license configuration file for FLEXlm licensed applications in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

`lmreread` – Instructs the FLEXlm license daemon to reread the license file

SYNOPSIS

`lmreread` [`-c` *license_file*]

IMPLEMENTATION

All supported platforms

DESCRIPTION

The `lmreread` command lets the system administrator instruct the flexible license manager (FLEXlm) license daemon to reread the FLEXlm license file, which can be useful if the data in the license file has changed. You can load the new data into the FLEXlm license daemon without shutting down and restarting it.

The `lmreread` command uses the *license_file* from the command line (or the default file if *license_file* is not specified) only to find the license daemon to send it the command to reread the license file. The license daemon always rereads the original file that it loaded. If you must change the path to the license file read by the license daemon, you must shut down the daemon and restart it with the new license file path.

If the server line node names or port numbers have been changed in the license file, you cannot use `lmreread`. In this case, you must shut down the daemon and restart it for the changes to take effect.

The `lmreread` command does not change any option information specified in an options file. If the new license file specifies a different options file, the information is ignored. If you changed your license file and must reread the options file, you must shut down the daemon by using the `lmdown`(1) command and restart it by using the `lmgrd`(1) command. In the FLEXlm v2.4 release and later, `lmreread` is either a link to or a copy of the `lmutil`(1) utility.

The `lmreread` command accepts the following option:

`-c` *license_file* Uses the specified *license_file* as input. If you omit the `-c` option, `lmreread` uses the `LM_LICENSE_FILE` environment variable to find the license file to use. If that environment variable is not set, `lmreread` uses the `/usr/local/flexlm/licenses/license.dat` file. If you omit the *license_file* argument, `lmreread` uses the `LM_LICENSE_FILE` environment variable. If that environment variable is not set, `lmreread` uses the `/usr/local/flexlm/licenses/license.dat` file.

SEE ALSO

`lmdown`(1) for information about shutting down all FLEXlm license daemons gracefully
`lmgrd`(1) for information about starting up FLEXlm license daemons
`lmutil`(1) for information about the core FLEXlm utility

`license.dat`(5) for information about the license configuration file for FLEXlm licensed applications in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

`lmstat` – Reports status of FLEXlm license daemons and feature usage

SYNOPSIS

`lmstat` [`-a`] [`-A`] [`-c` *license_file*] [`-f` *feature*] [`-l` *regular_expression*] [`-s` *server*] [`-S` *daemon*] [`-t` *timeout*]

IMPLEMENTATION

All supported platforms

DESCRIPTION

The `lmstat` command provides information about the status of the flexible license manager (FLEXlm) license daemon server nodes, vendor daemons, vendor features, and users of each feature. Optionally, you can qualify information by specific server nodes, vendor daemons, or features. In the FLEXlm v2.4 release and later, `lmstat` is either a link to or a copy of the `lmutil`(1) utility.

The `lmstat` command accepts the following options:

`-a` Displays all active users of all features. You should not use the `lmstat -a` command too often because if there are many active users, the `lmstat -a` command can generate a lot of network activity.

`-A` Lists all active licenses.

`-c` *license_file* Uses the specified license file as input. If you omit the `-c` option, `lmstat` uses the `LM_LICENSE_FILE` environment variable to find the license file to use. If that environment variable is not set, `lmstat` uses the `/usr/local/flexlm/licenses/license.dat` file. If you omit the *license_file* argument, `lmstat` uses the `LM_LICENSE_FILE` environment variable. If that environment variable is not set, `lmstat` uses the `/usr/local/flexlm/licenses/license.dat` file.

`-f` *feature* Lists all users of the specified feature.

`-l` *regular_expression* Lists all users of the features who match the specified regular expression.

`-s` *server* Displays the status of the specified server (server node).

`-S` *daemon* Lists all users of the specified daemon's features.

`-t` *timeout* Specifies the time-out interval (in seconds) during which daemons must complete their connections to each other. The default value is 10 seconds. If the daemons are being run on busy systems or a very heavily loaded network, you may want to use a larger value.

SEE ALSO

`lmgrd`(1) for information about starting up FLEXlm license daemons
`lmutil`(1) for information about the core FLEXlm utility

`license.dat`(5) for information about the license configuration file for FLEXlm licensed applications in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

lmutil – Core FLEXlm utility

SYNOPSIS

lmutil [-c *license_file*] *command*

IMPLEMENTATION

All supported platforms

DESCRIPTION

The lmutil utility is the core flexible license manager (FLEXlm) utility. Usually, end users will not use lmutil directly; they use the individual commands, which are either a copy of or a link to the lmutil utility.

The lmutil utility accepts the following arguments:

-c *license_file* Uses the specified *license_file* as input. If you omit the -c option, lmutil uses the LM_LICENSE_FILE environment variable to find the license file to use. If that environment variable is not set, lmutil uses the /usr/local/flexlm/licenses/license.dat file. If you omit the *license_file* argument, lmutil uses the LM_LICENSE_FILE environment variable. If that environment variable is not set, lmutil uses the /usr/local/flexlm/licenses/license.dat file.

command Links to the specified command. *command* may be lmcksum, lmdown, lmhostid, lmremove, lmreread, lmstat, or lmver.

SEE ALSO

lmcksum(1) for information about computing a checksum for the FLEXlm license file
lmdown(1) for information about shutting down all FLEXlm license daemons gracefully
lmhostid(1) for information about how to display the FLEXlm host ID of a system
lmremove(1) for information about removing specific FLEXlm licenses and returning them to the pool
lmreread(1) for information about instructing the FLEXlm license daemon to reread the FLEXlm license file
lmstat(1) for information about reporting status of FLEXlm license manager daemons and feature usage
lmver(1) for information about how to display the FLEXlm version being used

license.dat(5) for information about the license configuration file for FLEXlm licensed applications in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

`lmver` – Displays the FLEXlm version being used

SYNOPSIS

`lmver` *filename*

IMPLEMENTATION

All supported platforms

DESCRIPTION

The `lmver` command scans the contents of a binary or library file for the flexible license manager (FLEXlm) version string and displays it. On UNICOS systems, you must use the *filename* operand to display the correct FLEXlm version. If you omit the *filename* operand, `lmver` assumes the file name is `lmgr.a` and tries to find and display the version from the `lmgr.a` file. In the FLEXlm v2.4 release and later, `lmver` is either a link to or a copy of the `lmutil`(1) utility.

The `lmver` command accepts the following operand:

filename Specifies the file to scan to display the FLEXlm version being used. On UNICOS systems, you must use *filename*, and it must be `/usr/lib/libcraylm.a`.

SEE ALSO

`lmutil`(1) for information about the core FLEXlm utility

NAME

ln – Links files

SYNOPSIS

ln [-f] [-m] [-s] *file...* *target*

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4
AT&T extension (-s option)
CRI extension (-m option)

DESCRIPTION

The ln utility creates a link between *file* and *target*. A link is a directory entry that refers to a file; the same file may have several links to it. Under no circumstance can *file* and *target* be the same (take care when using sh(1) metacharacters).

If more than one *file* is specified, *target* must be an existing directory. If *target* is an existing directory, for each specified *file*, a link of the same name is created in the *target* directory. If *target* is not a directory, it is created as a link to *file*. If *target* is an existing file, it will not be overwritten unless the -f option is specified.

The ln utility accepts the following options and operands:

-f Forces files to be linked, even if the target exists. This option works only for hard links.

-m Creates a multilevel symbolic link.

-s Creates symbolic links.

file The path name of a file to be linked.

target The path name of the new link to be created, or the pathname of an existing directory in which the new links are to be created.

There are three kinds of links: hard links, symbolic links, and multilevel symbolic links.

A hard link (the default) can be made only to an existing file. To remove a file with more than one hard link, you must remove all links (including the name by which it was created). Hard links cannot span file systems or refer to directories.

A symbolic link contains the name of the file or directory to which it is linked. Symbolic links may span file systems and may refer to directories.

A multilevel symbolic link is a symbolic link that imposes a multilevel directory structure on any directory to which it points. It works much the same way a symbolic link works, with the exception that it causes the path name search to be deflected into a labeled subdirectory under the directory named in the symbolic link file. Creation of multilevel symbolic links is a privileged operation.

Your active security label must be equal to the file's security label (hard links only) and the parent directory for the new file entry (*target*) (unless the parent directory has a wildcard security level; then a file with any security level may be linked to the wildcard directory).

NOTES

If this utility is installed with a privilege assignment list (PAL), a user with one of the following active categories is allowed to perform the actions shown:

Active Category	Action
`system`, `secadm`	Allowed to link any file or directory.
`sysadm`	Allowed to link any file or directory, subject to security label restrictions. Shell-redirected I/O is subject to security label restrictions.

If the `PRIV_SU` configuration option is enabled, the super user is allowed to link any file or directory.

EXIT STATUS

The `ln` utility exits with one of the following values:

0 All the specified files were linked successfully.

>0 An error occurred.

EXAMPLES

Example 1: The following example links the existing file `example.c` to `ex.c`:

```
$ ln example.c ex.c
```

Example 2: The following example creates a symbolic link:

```
$ ln -s /usr/include incl
$ ls -l incl
lrwxrwxrwx  1  jtk      12 May 10 14:59 incl -> /usr/include
```

SEE ALSO

`cp`(1), `cpio`(1), `mv`(1), `rm`(1), `sh`(1)

`chmod`(2), `link`(2), `readlink`(2), `stat`(2), `symlink`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

General UNICOS System Administration, Cray Research publication SG–2301

NAME

`locale` – Gets locale-specific information

SYNOPSIS

`locale` [`-a` | `-m`]

`locale` [`-c`] [`-k`] *name* ...

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The `locale` utility writes information about the current locale environment, or all public locales, to the standard output. A *public* locale is one that is accessible to the application.

When you use `locale` without any arguments, it summarizes the current locale environment for each locale category determined by the settings of the `LC_CTYPE`, `LC_COLLATE`, `LC_TIME`, `LC_NUMERIC`, `LC_MONETARY`, and `LC_MESSAGES` environment variables

When you specify a keyword *name*, `locale` selects the named keyword and the category containing that keyword. When a category *name* is specified, `locale` selects the named category and all keywords in that category.

The `locale` utility accepts the following options and operands:

`-a` Writes information about all available public locales. The available locales include `POSIX`, which represents the POSIX locale.

`-m` Writes names of available charmaps.

`-c` Writes the names of selected locale categories.

`-k` Writes the names and values of selected keywords.

name The name of a locale category, the name of a keyword in a locale category, or the reserved name `charmap`. The specified category or keyword is selected for output. You can specify both category and keyword names as *name* operands, in any sequence.

NOTES

If this utility is installed with the default privilege assignment list (PAL), a user with an active `secadm` or `sysadm` category may override mandatory write access protections on a file, any directory in the file path, or, in a privileged administrator shell environment, any file to which input or from which output is being redirected.

EXAMPLES

In the following examples, the assumption is that locale environment variables are set as follows:

```
LANG=locale_x
LC_COLLATE=locale_y
```

Example 1: Invoking `locale` with no arguments results in the following:

```
$ locale
LANG=locale_x
LC_CTYPE="locale_x"
LC_COLLATE=locale_y
LC_TIME="locale_x"
LC_NUMERIC="locale_x"
LC_MONETARY="locale_x"
LC_MESSAGES="locale_x"
LC_ALL=
```

Example 2: Set the `LC_ALL` environment variable to the `POSIX` locale and print out the value of the keyword `decimal_point`:

```
$ LC_ALL=POSIX locale -ck decimal_point
LC_NUMERIC
decimal_point="."
```

EXIT STATUS

The `locale` utility exits with one of the following values:

0 All the requested information was found and output successfully.

>0 An error occurred.

SEE ALSO

`localedef`(1)

General UNICOS System Administration, Cray Research publication SG–2301

NAME

`localedef` – Defines locale environment

SYNOPSIS

`localedef` [`-c`] [`-f` *charmap*] [`-i` *sourcefile*] *name*

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The `localedef` utility converts source definitions for locale categories into a format usable by the functions and utilities whose operational behavior is determined by the setting of the locale environment variables.

The `localedef` utility reads source definitions for one or more locale categories that belong to the same locale from the file specified in the `-i` option (if specified) or from standard output.

Each category source definition is identified by the corresponding environment variable name and terminated by an `END` *category-name* statement. The following categories are supported.

`LC_COLLATE` Defines collation rules.

`LC_CTYPE` Defines character classification and case conversion.

`LC_MESSAGES` Defines the format and values of affirmative and negative responses.

`LC_MONETARY` Defines the format and symbols used in formatting of monetary information.

`LC_NUMERIC` Defines the decimal delimiter, grouping, and grouping symbol for nonmonetary numeric editing.

`LC_TIME` Defines the format and content of date and time information.

The `localedef` utility accepts the following options and operands:

`-c` Creates permanent output even if warning messages have been issued.

`-f` *charmap* Specifies the path name of a file that contains a mapping of character symbols and collating element symbols to actual character encodings. If symbolic names (other than collating symbols defined in a `collating-symbol` keyword) are used, you should specify this option.

`-i` *sourcefile* Specifies the path name of a file that containis the source definitions.

name Identifies the target locale. The utility supports the creation of *public*, or generally accessible locales, as well as *private*, or restricted access locales. If the name contains one or more slash (/) characters, *name* is interpreted as a pathname where the created locale definition(s) will be stored. If the name does not contain any slash characters, the locale will be *public*. The ability to create *public* locales is restricted to users with appropriate privileges. If you omit this option, source definitions are read from standard input.

EXIT STATUS

The `localedef` utility exits with the following value:

3 The capability to create new locales is not supported.

SEE ALSO

`locale`(1)

NAME

logger – Makes entries in the system log

SYNOPSIS

logger [-d] [-f *file*] [-h *host*] [-i] [-l *logname*] [-p *pri*] [-t *tag*] [-P *port*] [-T] [*messages*]

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4
AT&T extensions (-f, -i, -p, and -t options)
CRI extensions (-d, -h, -l, -P, and -T options)

DESCRIPTION

The logger utility provides a program interface to the syslog(3C) system log routine. A message can be given on the command line, which is logged immediately, or a file is read and each line is logged.

The logger utility accepts the following options:

-d Opens the pipe to syslogd(8) without the O_NDELAY flag.

-f *file* Logs the specified file.

-h *host* Sends message to the daemon on the remote host, rather than the daemon on the local machine.

-i Logs the process ID of the log process with each line.

-l *logname* Alternates name for the named pipe interface to the syslog(3C) daemon.

-p *pri* Enters the message with the specified priority (*pri*). The priority may be a number from 0 through 7, or a corresponding word, as follows:

```
0       emerg
1       alert
2       crit
3       err
4       warning
5       notice
6       info
7       debug
```

pri may be preceded by a facility, in the form *facility*.*pri*. Valid facilities include the following:

```
kern            user            mail
daemon          auth            syslog
lpr             local0          local1
local2          local3          local4
local5          local6          local7
```

For a discussion of these facilities, see `syslog`(3C). For example, `-p daemon.info` logs messages as informational (level 6) in the daemon facility. The default is `user.notice`.

`-t` *tag* Marks every line in the log with the specified *tag*.

`-P` *port* Uses the specified TCP/IP port, rather than the one specified in `/etc/services`.

`-T` Uses TCP/IP socket, rather than the named pipe interface to the local `syslog`(3C) daemon.

messages The messages to log. If not specified, the file specified with `-f` or standard input is logged.

EXIT STATUS

The `logger` utility exits with one of the following values:

0 Successful completion.

>0 An error occurred.

EXAMPLES

Example 1: If you restart a daemon in the middle of the day, you could log this event with the following command:

```
$ logger -p user.info restarted development copy of syslog daemon
```

The message is as follows:

```
restarted development copy of syslog daemon
```

Example 2: If you are a system operator and have cleaned out files in `/tmp`, you could log this event in the system log maintained by `syslogd`(8), using the following command:

```
$ logger /tmp was cleaned out by hand when it filled up.
Joe Operator
```

The message is as follows:

```
/tmp was cleaned out by hand when it filled up.
Joe Operator
```

SEE ALSO

`syslog`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`log`(4) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

`syslogd`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

login – Signs on

SYNOPSIS

login [*name* -L *requested_label* [*env-vars*]]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The login utility is used at the beginning of each terminal session and lets you identify yourself to the system. It is invoked 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 pressing <CONTROL-d> to indicate an end-of-file.

The login utility invokes the centralized identification and authorization library routines to validate the user ID and password.

login accepts the following options:

name Your login name.

-L *requested_label*
Security label with which you want to log in. The format of the requested label is *level*[,*compartment*[,*compartment*]]. The *requested_label* must immediately follow the *name*.

env-vars Environment variable that you can set.

The login command requests 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.

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)) are initialized, and the file .profile in the working directory is executed, if it exists. In addition, if executing in a secure UNICOS environment, the user's default and maximum class, default and authorized categories, default and authorized compartments, minimum and maximum security levels, default security level, and security permissions are set. These specifications are found in the user's /etc/udb file entry. The name of the command interpreter is - followed by the last component of the interpreter's path name (that is, -sh). If this field in the udb file is empty, the default command interpreter, /bin/sh, is used. If this field is *, a chroot(2) is performed to the directory named in the directory field of the entry. At that point, login is reexecuted at the new level, which must have its own root structure, including /bin/login and /etc/udb.

The basic environment (see sh(1)) is initialized to the following:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin:/usr/ucb
SHELL=*last-field-of-udb-ue_shell field*
MAIL=/usr/mail/*your-login-name*
LOGNAME=*your-login-name*

You may modify the environment by supplying additional arguments to login, either at execution time or when login requests your login name. The *env-vars* arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as follows:

L*n*=xxx

The *n* argument is a number starting at 0 and is incremented each time a new variable name is required. Variables containing = are placed into the environment without modification. If they already appear in the environment, they replace the older value. There are two exceptions: the variables PATH and SHELL cannot be changed. Users logging into restricted shell environments are thus prevented from spawning secondary shells that are not restricted.

The login utility understands simple single-character quoting conventions. Typing a backslash (\) in front of a character quotes it and allows the inclusion of such characters as spaces and tabs.

NOTES

It is not possible to exec(2) login from a normal user ID.

The number of unsuccessful login attempts that will be allowed before login terminates is configurable. This parameter is set in the /etc/config/confval file with the following line:

```
login.login_attempts:   "3"
```

"3" is the number of unsuccessful login attempts allowed. This line is retrieved with the getconfval(3C) library call. If this line does not exist (the default) or the number is set to 0, no limit will be placed on the number of unsuccessful login attempts.

If the IP security option is not enabled, login requests are validated to ensure that the remote host or workstation's security levels and compartments as defined in the /etc/config/spnet.conf file are included in the security level range and authorized compartment range for the UNICOS system. Your security values are set to the most restrictive boundary conditions as defined by the network access list (NAL) and the user database (UDB).

If the IP security option is enabled, the checks for the /etc/config/spnet.conf file are done by the kernel. The socket's security label is set by the kernel when this check is done. login sets the user's security label to the most restrictive boundary conditions as defined by the socket's security label (as defined by the NAL) and the UDB.

Your active security level and active compartments are obtained from the dev/ttyp file, which contains the security label present with an IP security option. If dev/ttyp has a null security label, the user session is restricted to operating with a null security label, and you are not allowed to change the active security level and active compartments set by login.

The login process also validates your right to access the UNICOS system from the host or workstation issuing the login request. Access to UNICOS is granted or denied based upon the workstation access list (WAL) check performed by the login process.

You cannot log in if you exceed the MAXLOGS setting.

The results of user validation are recorded in the security log.

If this utility is installed with a privilege assignment list (PAL), a user with one of the following active categories is allowed to perform the actions shown:

Active Category	Action
system	Allowed to use this utility.

If the PRIV_SU configuration option is enabled, the super user is allowed to use this utility.

MESSAGES

Login incorrect
: Your name and the password may not match. This is a generic message for any of several login failure causes. No more information is given. The user name may be invalid or the password may be wrong. You may have been denied access by the WAL check. Your login can be locked, disabled, or invalidated for security violations.

No shell
Bad account ID
Bad group ID list
Bad user ID
Unable to change to login root
No Root Directory
: Account may not be set up correctly; consult a system administrator.

No utmp entry
: You must execute login from the lowest-level shell. You attempted to execute login as a command without using the shell's exec internal command or from other than the initial shell.

Unable to give N shares to user
: The setshares call failed. Contact your system support staff.

Unable to create new job
: The setjob call failed. Contact your system support staff.

Unable to make job temporary directory
: The makejtmp call failed. Contact your system support staff.

`Unable to lock the UDB`
: The `lockudb` call failed. Contact your system support staff.

`Unable to update the UDB`
: The rewrite `udb` call failed while attempting to write the last login record. Contact your system support staff.

`Lastlog update error`
: Unable to reread the `udb` entry to write the last login record. Contact your system support staff.

`Login not allowed at this node`
: You are not allowed to log in at this network node for security reasons. Use an authorized terminal for logging in.

`Unable to get host by name`
: The host name that accompanies the login request is not defined in `/etc/hosts` or is *null*.

`Could not access NAL`
: An error was detected when `/etc/config/spnet.conf` was opened. Contact your system support staff.

`No login without NAL entry`
: The user does not have an NAL entry for this remote node.

`Can't set default security level`

`Can't set default security compartments`

`urm: job exceeds memory maximum`
: The job would exceed the memory oversubscription amount configured in the Unified Resource Manager (URM).

`urm: job exceeds job maximum`
: The job would exceed the number of allowed jobs configured in the Unified Resource Manager (URM).

`Invalid requested label`
: The requested label is not valid.

FILES

`/bin/passwd`	Program that changes passwords
`/bin/sh`	Standard shell
`/dev/tty*`	Login devices
`/etc/dialups`	List of devices that need a dial-up password
`/etc/d_passwd`	Dial-up passwords for `/etc/dialups`
`/etc/utmp`	Accounting file

`/etc/wtmp`	Accounting file
`/usr/mail/$LOGNAME`	Mailbox for account `$LOGNAME`
`/etc/udb`	User validation file containing user control limits
`/etc/config/confval`	Number of bad login attempts after which `login` terminates
`/etc/config/spnet.conf`	Network access list (NAL) and workstation access list (WAL)

SEE ALSO

`mail`(1), `passwd`(1), `sh`(1), `su`(1)

`chroot`(2), `exec`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

`getconfval`(3C), `ia_failure`(3C), `ia_mlsuser`(3C), `ia_success`(3C), `ia_user`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`cshrc`(5), `profile`(5), `udb`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

`chroot`(8), `getty`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

`checkwal`(), `fetchnal`() (library routines in `/libc/gen`)

General UNICOS System Administration, Cray Research publication SG–2301

NAME

logname – Gets user's login name

SYNOPSIS

logname

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The logname utility uses getlogin(3C) to find the login name of the user and writes that name to standard output.

EXIT STATUS

The logname utility exits with one of the following values:

0 Successful completion.

>0 An error occurred.

SEE ALSO

env(1), login(1), sh(1)

getlogin(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

`lorder` – Finds ordering relation for an object library

SYNOPSIS

`lorder` *files*

IMPLEMENTATION

All Cray Research systems

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 or archive member 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).

Link editor `ld`(1) is capable of multiple passes over an archive in the portable archive format (see `ar`(5)) and does not require that `lorder` be used when building an archive. The usage of the `lorder` utility may, however, allow for a slightly more efficient access of the archive during the link edit process. The use of this utility is not recommended.

The `lorder` utility accepts the following option:

files Names of object or library archive files you specify.

NOTES

When given a nonexistent file, `lorder` returns an exit status of 0.

WARNINGS

The `lorder` utility accepts as input any object or archive file, regardless of its suffix, provided that there is more than one input file. If there is only one input file, its suffix must be `.o`.

EXAMPLES

The following example builds a new library from existing `.o` files:

```
bar cr library ‘lorder *.o | tsort‘
```

FILES

`TMPDIR/*symdef` Temporary file

`TMPDIR/*symref` Temporary file

SEE ALSO

`ar`(1), `ld`(1), `tsort`(1)

`tmpnam`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`ar`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

`tsar`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

lp – Sends files to a printer

SYNOPSIS

lp [-c] [-d *dest*] [-n *copies*] [-s] [*file...*]

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The lp utility copies the input files to an output device. The actual writing to the output device occurs after the lp utility successfully exits.

The lp utility accepts the following options:

-c Exits only after further access to any of the input files is no longer required. The application can then safely delete or modify the files without affecting the output operation.

-d *dest* Specifies a string that specifies the output device or destination. The -d *dest* option takes precedence over the LPDEST environment variable, which in turn takes precedence over the PRINTER environment variable.

-n *copies* Writes *copies* number of copies of the files; *copies* is a positive decimal integer.

-s Suppress printing Request id is . . . message to standard output.

file Denotes a path name of a file to be output. If you omit *file* operands, or if a *file* operand is -, the standard input is used. If a *file* operand is used, but you omit -c option, the process performing the writing to the output device may have user and group permissions that differ from that of the process invoking lp.

ENVIRONMENT VARIABLES

LPDEST This variable is interpreted as a string that names the output device or destination. If the LPDEST environment variable is not set, the PRINTER environment variable is used. The -d *dest* option takes precedence over LPDEST.

PRINTER This variable is interpreted as a string that names the output device or destination. If the LPDEST and PRINTER environment variables are not set, a system default printer is used. The -d *dest* and the LPDEST environment variable take precedence over PRINTER.

EXIT STATUS

The `lp` utility exits with one of the following values:

0 All input files were processed successfully.

>0 No output device was available, or an error occurred.

SEE ALSO

`lpr`(1B), `sh`(1)

NAME

lpq – Spool queue examination program

SYNOPSIS

/usr/ucb/lpq [+[*n*]] [-l] [-P*printer*] [*jobnums*] [*users*]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The lpq utility examines the spooling area used by lpd(8) for printing files on the line printer, and reports the status of the specified jobs or all jobs associated with a user. If you invoke lpq without any arguments, it reports on any jobs currently in the queue. The lpq utility prints only those jobs at the current security label.

The lpq utility accepts the following options:

+[*n*] Displays the spool queue. When *n* (a number) is specified, lpq sleeps for *n* seconds between scans of queue.

-l Causes information for each file comprising a job to be printed.

-P*printer* Specifies a particular printer. When P is not specified, either the default line printer or the value of the PRINTER variable in the environment is used.

jobnums Specifies job numbers that should be examined.

users Specifies user names that should be examined.

For each job submitted (that is, each invocation of lpr(1B)), lpq reports the user's name, the current rank in the queue, the names of files that compose the job, the job identifier (a number that may be supplied to lprm(1B) for removing a specific job), and the total size (in bytes). Unless the -l option is used, only as much information as will fit on one line is displayed. Job ordering depends on the algorithm used to scan the spooling directory and is supposed to be FIFO (first-in, first-out). File names composing a job may be unavailable (when lpq is used as a sink in a pipeline), in which case, the file is indicated as standard input.

If lpq warns that no daemon present (that is, because of some malfunction), the lpc(8) command can be used to restart the printer daemon.

NOTES

If users try to remove files other than their own, permission will be denied.

If this utility is installed with a privilege assignment list (PAL), a user who is assigned the following privilege text upon execution of this command is allowed to perform the action shown:

Privilege Text	Action
admin	Allowed to see the status of all jobs in a print queue. Other users may only see jobs in the print queue at the same label as the user.

If this command is installed with a PAL, a user with one of the following active categories is allowed to perform the action shown:

Active Category	Action
sysadm, sysadm	Allowed to see the status of all jobs in a print queue.

If the PRIV_SU option is enabled, the super user is allowed to see all jobs in the print queue.

BUGS

The output of lpq may be somewhat unreliable because of the dynamic nature of the information in the spooling directory.

Output formatting is sensitive to the line length of the terminal; this can result in widely spaced columns.

The lpq utility is sometimes unable to open various files because the lock file is malformed.

FILES

/etc/printcap	To determine printer characteristics
/etc/termcap	To manipulate the screen for repeated display
/usr/spool/*	Spooling directory, as determined from printcap
/usr/spool/*/cf*	Control files specifying jobs
/usr/spool/*/lock	Lock file to obtain the currently active job

SEE ALSO

lpr(1B), lprm(1B)

printcap(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

lpc(8), lpd(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

lpr – Prints off-line

SYNOPSIS

/usr/ucb/lpr [-P*printer*] [-#*num*] [-C *class*] [-J *job*] [-T *title*] [-R] [-i[*numcols*]] [-1|2|3|4*font*] [-w*num*] [-p] [-l] [-t] [-n] [-d] [-g] [-v] [-c] [-f] [-r] [-m] [-h] [-s] [*files*]

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The lpr utility uses a spooling daemon to print the files specified by *files* when facilities become available. If no files are specified, standard input is assumed.

The lpr utility accepts the following option to specify a printer:

-P*printer* Forces output to a specific printer. Usually, the default printer is used (site-dependent), or the value of the PRINTER environment variable is used.

The following single-letter options are used to notify the line printer spooler that the files are not standard text files. The spooling daemon uses the appropriate filters to print the data accordingly.

-p Uses pr(1) to format the files (equivalent to print).

-l Uses a filter that allows control characters to be printed and suppresses page breaks.

-t Assumes that the files contain data from troff (cat phototypesetter commands).

-n Assumes that the files contain data from ditroff (device-independent troff).

-d Assumes that the files contain data from tex (DVI format from Stanford).

-g Assumes that the files contain standard plot data as produced by the plot routines (for the filters used by the printer spooler).

-v Assumes that the files contain a raster image for devices such as the Benson Varian.

-c Assumes that the files contain data produced by cifplot.

-f Uses a filter that interprets the first character of each line as a standard Fortran carriage control character.

You can use the following options:

`-#`*num* Prints multiple copies of output; *num* is the number of copies desired of each file specified. For example, the following command line would result in three copies of file `foo.c`, followed by three copies of file `bar.c`, and so on.

```
lpr -#3 foo.c bar.c more.c
```

On the other hand, the following command line would produce three copies of the concatenation of the files:

```
cat foo.c bar.c more.c | lpr -#3
```

`-C` *class* Uses *class* arguments as a job classification for use on the burst page. For example, the following command line would cause the system name (the name returned by `hostname`(1)) to be replaced on the burst page by `EECS`, and file `foo.c` to be printed:

```
lpr -C EECS foo.c
```

`-J` *job* Uses *job* as the job name to print on the burst page. Usually, the name of the first file is used.

`-T` *title* Uses *title* as the title used by `pr`(1), instead of the file name.

`-R` Writes a message to standard output containing the unique number which is used to identify this job. This number can be used to cancel (see `lprm`(1B)) or find the status (see `lpq`(1B)) of the job.

`-i`[*numcols*]
Indents the output. If *numcols* is numeric, it will be used as the number of blanks to be printed before each line; otherwise, 8 characters are printed.

`-1|2|3|4`*font*
Specifies a font to be mounted on font position 1, 2, 3, or 4. The daemon constructs a `.railmag` file referencing file `/usr/lib/vfont/name.size`.

`-w`*num* Sets the page width to *num*.

`-r` Removes the file on completion of spooling or on completion of printing (with the `-s` option).

`-m` Sends mail on completion.

`-h` Suppresses the printing of the burst page.

`-s` Links data files rather than trying to copy them so that large files can be printed.

files Files to be printed. This means that the files must not be modified or removed until they have been printed.

NOTES

If you try to spool a file that is too large, it will be truncated.

lpr objects to printing binary files.

If a user other than root prints a file and spooling is disabled, lpr will print a message saying this and will not put jobs in the queue.

If a connection to lpd(8) on the local machine cannot be made, lpr will indicate that the daemon cannot be started. Messages can be printed in the daemon's log file regarding missing lpd spool files.

BUGS

Fonts for troff and tex reside on the host with the printer. Currently, local font libraries cannot be used.

FILES

/etc/printcap	Printer capabilities database
/etc/udb	User validation file containing user control limits
/usr/lib/lpd*	Line printer daemons
/usr/lib/lpf	Sample lpr output filtering program
/usr/lib/necf	Sample lpr output filtering program
/usr/spool/*	Directories used for spooling
/usr/spool/*/cf*	Daemon control files
/usr/spool/*/df*	Data files specified in cf files
/usr/spool/*/tf*	Temporary copies of cf files

SEE ALSO

lp(1), lpq(1B), lprm(1B), pr(1)

lpc(8), lpd(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

`lprm` – Removes jobs from the line printer spooling queue

SYNOPSIS

`/usr/ucb/lprm` [`-P`*printer*] [`-`] [*jobnums*] [*users*]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `lprm` utility removes a job, or jobs, from a printer's spool queue. Because the spooling directory is protected from users, using `lprm` is typically the only method by which you may remove a job.

If you specify `lprm` without any arguments, it deletes the currently active job if it is owned by the user who invoked `lprm`.

The `lprm` utility accepts the following options:

`-P`*printer* Specifies the queue associated with a specific printer; otherwise, the default printer or the value of the `PRINTER` environment variable is used.

– Removes all jobs that a user owns. If the super user uses this option, the spool queue will be emptied entirely.

jobnums Used to dequeue an individual job.

users Removes any jobs queued belonging to the user (or users).

You may dequeue an individual job by specifying its job number. This number may be obtained from the `lpq`(1B) program. For example, if you use the following program, `lprm` will announce the names of any files it removes; it will remain silent if there are no jobs in the queue that match the request list:

```
$ lpq -l
1st: ken        [job #013ucbarpa]
        (standard input)        100 bytes
$ lprm 13
```

The `lprm` utility kills an active daemon, if necessary, before removing any spooling files. If a daemon is killed, a new one is automatically restarted upon completion of file removal.

NOTES

If users try to remove files other than their own, permission will be denied.

If this utility is installed with a privilege assignment list (PAL), a user who is assigned the following privilege text upon execution of this command is allowed to perform the action shown:

Privilege Text	Action
admin	Allowed to remove any job in a print queue. Other users may only remove their own jobs.

If this command is installed with a PAL, a user with one of the following active categories is allowed to perform the action shown:

Active Category	Action
sysadm, sysadm	Allowed to remove any job in a print queue.

If the PRIV_SU option is enabled, the super user is allowed to remove any job in a print queue.

BUGS

Because race conditions are possible in the update of the lock file, the currently active job may be incorrectly identified.

FILES

/usr/spool/*	Spooling directories
/etc/printcap	Printer characteristics file
/usr/spool/*/lock	Lock file used to obtain the process identification number (PID) of the current daemon and the job number of the currently active job

SEE ALSO

lpq(1B), lpr(1B)

NAME

`ls` – Lists contents of directory

SYNOPSIS

`ls` [`-1`] [`-a`] [`-A`] [`-b`] [`-B`] [`-c`] [`-C`] [`-d`] [`-e`] [`-f`] [`-F`] [`-g`] [`-i`] [`-k`] [`-I`] [`-l`] [`-L`] [`-m`] [`-n`] [`-o`] [`-p`] [`-P`] [`-q`] [`-r`] [`-R`] [`-s`] [`-t`] [`-u`] [`-x`] [*file* ...]

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4
AT&T extensions (`-b` option)
CRI extensions (`-A`, `-B`, `-e`, `-k`, `-l`, and `-P` options)

DESCRIPTION

For each directory argument, `ls` lists the contents of the directory; for each file argument, `ls` repeats the file name and any other information requested. By default, the output is sorted alphabetically. When no argument is specified, the current directory is listed. When several arguments are specified, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format for output directed to a terminal is multi-column with entries sorted down the columns. The `-1` option allows single-column output, and the `-m` option enables stream output format in which files are listed across the page, separated by commas.

To determine output formats for the `-C`, `-x`, and `-m` options, `ls` uses the `COLUMNS` environment variable, which determines the number of character positions available on one output line. If this variable is not set, the system is prompted by using the `ioctl`(2) system call to acquire the current window size. When this information cannot be obtained, 80 columns are assumed.

Use the `-e` option to display security information.

The `ls` utility accepts the following options:

`-1` (Number one) Forces output format of one entry per line. This is the default format when the output is not directed to a terminal.

`-a` Lists all entries, including entries whose names begin with a dot (`.`), which usually are not listed.

`-A` Lists all entries, including entries whose names begin with a dot (`.`), except for the dot (`.`) and dot-dot (`..`) files.

`-b` Forces the printing of nongraphic characters to be in the octal *ddd* notation.

`-B` Writes the number of file system blocks a file occupies, as specified by the `-l` and `-s` options, in 4096-byte units, rather than the default 512-byte units.

`-c` Uses time of last modification of the inode (file created, mode changed, and so on) for sorting (`-t`) or printing (`-l`).

`-C` Specifies multicolumn output with entries sorted down the columns. This is the default format when the output is directed to a terminal.

`-d` If the *files* argument is a directory, lists only its name (not its contents); often used with `-l` to get the status of a directory.

`-e` Provides the access control list (ACL) flag, integrity label flag, security level, and compartment flag as the last fields before the file name, or, when used with `-l`, as the fields immediately following the mode field. When at least one file compartment is set, the file's compartment flag is displayed as a plus sign (`+`) adjacent to the file's security level. An `i` indicates that the file has an integrity class or category assigned to it; ask your security administrator to relabel these values to 0. When a file has an associated ACL, its ACL flag appears as an `a` adjacent to the mode field (if present), or following any other selected fields. A file or directory that has a wildcard security level has an asterisk (`*`) displayed for its file level. A trusted facility management file or directory has a `T` displayed for its file level; if this symbol is displayed, ask your security administrator to relabel the file with the proper security label. The file's security level is displayed as a question mark (`?`) if it is outside the user's allowable range, or if any of the file's compartments are not in the user's set of valid compartments. No security information is displayed when a question mark is shown.

`-f` Forces each argument to be interpreted as a directory and lists the name found in each slot. This option turns off `-A`, `-l`, `-t`, `-s`, and `-r`, and turns on `-a`; the order is the order in which entries appear in the directory.

`-F` Adds a slash (`/`) after each file name that is a directory, adds an asterisk (`*`) after each file name that is executable, adds an at sign (`@`) after each file name that is a symbolic link, adds an equals sign (`=`) after each file name that is a socket, and a vertical line (`|`) after each file name that is a FIFO.

`-g` The same as `-l`, except that the owner is not printed.

`-i` For each file, prints the inode number in the first column of the report.

`-k` Writes the number of file system blocks a file occupies, as specified by the `-l` and `-s` options, in 1024-byte units, rather than the default 512-byte units.

`-I` Same as `-i`, but the inode is printed as two 32-bit values.

`-l` Lists in long format, specifying mode, number of links, owner, group, size in bytes, and time of last modification for each file. See the EXAMPLES section. If the file is a special file, the size field will contain the major and minor device numbers rather than a size. If the file is a symbolic link, the path name of the linked-to file will be printed, preceded by `->`; or, if the file is a multilevel symbolic link and the `-e` option is used, the pathname will be preceded by `*>`. If *file* is a directory, each list of files within the directory is preceded by a status line that indicates the number of file system blocks occupied by files in the directory in 512-byte units (rounded up, if necessary).

`-L` If an argument is a symbolic link, lists the file or directory the link references rather than the link itself.

`-m` Specifies stream output format.

`-n` The same as `-l`, except that the owner's UID and GID numbers (and account ID numbers for `-P`), rather than the associated character strings, are printed.

`-o` The same as `-l`, except that the group is not printed.

`-p` Puts a slash (`/`) after each file name if that file is a directory.

`-P` Lists the account ID associated with each file.

`-q` Forces printing of nongraphic characters in file names as the character `?`.

`-r` Reverses the order of sort to get reverse alphabetic or oldest first, as appropriate.

`-R` Recursively lists subdirectories encountered.

`-s` Gives size of each file in terms of file system blocks. The size is given in 512-byte units by default, and may be changed by the `-k` or `-B` options.

`-t` Sorts by time modified (latest first) rather than by name.

`-u` Uses time of last access rather than last modification for sorting (with the `-t` option) or printing (with the `-l` option).

`-x` Specifies multi-column output with entries sorted across rather than down the page.

files Files to be listed. If no *files* are specified, all files in the current directories are listed.

The mode printed under the `-l` option consists of 10 characters. The first character is one of the following:

`b` Block special file
`c` Character special file
`d` Directory.
`l` Symbolic link
`m` Migrated file
`p` FIFO (named pipe) special file
`R` Restart file; for more information on the restartability of core files, see `restart`(1) and `core`(5).
`s` Type socket
`-` Ordinary file

The next 9 characters are interpreted as three sets of 3 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 as permission to search the directory for a specified file. The permissions are indicated as follows:

`r` The file is readable.
`w` The file is writable.
`x` The file is executable.

- `-` The indicated permission is not granted.
- `l` Mandatory locking occurs during access (the set-group-ID it is on, and the group execution bit is off).
- `t` Sets the sticky bit (see `chmod`(2) for more information). Only the super user or owner of the directory can alter the `t` permission (mode 1000).
- `s` The set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on.
- `S` Undefined bit-state (the set-user-ID bit is on and the user execution bit is off).

For user and group permissions, the third position is sometimes occupied by a character other than `x` or `-`. `s` may also occupy this position; it refers to the state of the set-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as `root` but must assume the identity of the user stated at login.

In the case of the sequence of group permissions, `l` may occupy the third position. `l` refers to mandatory file and record locking. This permission describes a file's ability to allow other processes to lock its reading or writing permissions during access.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

If you specify the `-l` option, each list of files within the directory is preceded by a status line indicating the number of file system blocks occupied by files in the directory in 512-byte units.

NOTES

If this utility is installed with a privilege assignment list (PAL), a user with one of the following active categories is allowed to perform the actions shown:

Active Category	Action
`system`, `secadm`	In a privileged administrator shell environment, shell-redirected I/O is not subject to file protections.
`sysadm`	Shell-redirected output is subject to security label restrictions.

If the `PRIV_SU` configuration option is enabled, shell-redirected I/O on behalf of the super user is not subject to file protections.

EXIT STATUS

The `ls` utility exits with one of the following values:

0 All files were written successfully.

>0 An error occurred.

BUGS

Unprintable characters in file names may confuse the columnar output options.

In file names, <newline> and <tab> are considered printing characters.

The output device is assumed to be 80 columns wide.

If hard links are among the files, the total block count will be incorrect.

EXAMPLES

Example 1: The following is an example of output from the ls utility, using the -l option:

```
total 155
drwxr-xr-x   2 cray     os          96  Sep 23  12:42  onea
-rw-r--r--   1 cray     os        3380  Oct  3  08:18  save.ftpl
-rwx------   1 cray     os       74912  Sep 19  09:04  testx
```

Example 2: The following is an example of output from the ls utility using the -le options. The second column that follows the mode field shows the file's security level. The plus sign (+) appended to the security level of the second file indicates that the file has compartments. The third file has an associated ACL, as indicated by the a in column 1 adjacent to the mode field.

```
total 155
drwxr-xr-x     1    2 cray   os      96   Sep 23   12:42   onea
-rw-r--r--     1+   1 cray   os    3380   Oct 03   08:18   save.ftpl
-rwx------a    1    1 cray   os   74912   Sep 19   09:04   testx
```

FILES

/etc/udb	User validation file that contains user control limits
/etc/group	Group file that contains group names and group IDs

SEE ALSO

chmod(1), find(1), restart(1), chmod(2), socket(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

core(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

m4 – Invokes a macro processor

SYNOPSIS

m4 [-B *int*] [-D *name*[=*val*]] [-e] [-H *int*] [-S *int*] [-s] [-T *int*] [-U *name*] [*files*]

IMPLEMENTATION

All Cray Research systems

STANDARDS

XPG4

DESCRIPTION

The m4 utility invokes a macro processor intended for use as a general-purpose front end for any programming language. Each of the argument files is processed in order; if there are no files, or if a file name is -, standard input is read. The processed text is written on standard output.

The m4 command supports the following options:

-B *int*	Changes the size of the push-back and argument collection buffers from the default of 4096.
-D *name*[=*val*]	Defines *name* to *val* or to null in the absence of *val*.
-e	Operates interactively. Interrupts are ignored and the output is unbuffered.
-H *int*	Changes the size of the symbol table hash array from the default of 199. The size should be a prime number.
-S *int*	Changes the size of the call stack from the default of 100 slots.
-s	Enables line sync output for the C preprocessor (# "line . . .").
-T *int*	Changes the size of the token buffer from the default of 512 bytes.
-U *name*	Undefines *name*. Macros take 3 slots, and nonmacro arguments take 1.
files	Specifies the files to be processed.

Macro calls have the following form:

name(*arg*1,*arg*2, . . ., *argn*)

A (character must immediately follow the name of the macro. When the name of a defined macro is not followed by (, it is considered a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscores; the first character cannot be a digit.

Leading unquoted blanks, tabs, and new-line characters are ignored during the collection of arguments. Left single quotation marks (grave accent, ASCII 96) and right single quotation marks are used to quote strings. The value of a quoted string is the string stripped of the quotation marks.

When a macro name is recognized, its arguments are collected by a search for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments will be considered null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses that appear in 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 into the input stream and rescanned.

The `m4` command has the following built-in macros. They may be redefined, but after redefinition 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; `$*` is replaced by a list of all arguments separated by commas; `$@` is like `$*`, but each argument is quoted (with the current quotation characters).

`undefine` Removes the definition of the macro named in its argument.

`defn` Returns the quoted definition of its arguments. This is useful for renaming macros, especially built-in macros.

`pushdef` Functions as does `define`, but saves any previous definition.

`popdef` Removes current definition of its arguments, exposing the previous argument if one exists.

`ifdef` If the first argument is defined, the value will be the second argument; otherwise it will be the third. If there is no third argument, the value will be null. The words `unix` and `CRAY` are predefined on all UNICOS systems. The word `CRAYYMP` is predefined.

`shift` Returns all but its first argument. The other arguments are quoted and pushed back, with commas to separate them. The quoting nullifies the effect of the extra scan that will subsequently be performed.

`changequote` Changes quote symbols to the first and second arguments. The symbols may be up to 5 characters long. `changequote` without arguments restores the original values (that is, ‘ ’).

`changecom` Changes left and right comment markers from the default # and new-line character. Without arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes a new-line character. With two arguments, both markers are affected. Comment markers may be up to 5 characters long.

`divert`	Changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded. The final output is the concatenation of the streams in numeric order; initially stream 0 is the current stream. The `divert` macro maintains 10 output streams, numbered 0 through 9.
`undivert`	Causes immediate output of text from diversions named as arguments, or from all diversions if no argument exists. 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 character.
`ifelse`	Has three or more arguments. If the first argument is the same string as the second, the value will be the third argument. If not, and if there are more than four arguments, the process will be repeated with arguments four, five, six, and seven. Otherwise, the value will be 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 the interpreting of 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 `+`, `-`, `*`, `/`, `%`, bitwise `&`, `|`, `^`, and `~`; relationals; and parentheses. Octal and hexadecimal 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 at which 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`	Functions as does `include`, except that it returns nothing if the file is inaccessible.
`syscmd`	Executes the UNICOS command given in the first argument. No value is returned.
`sysval`	Returns 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 provided, is the exit code; the default is 0.

`m4wrap`	Pushes back argument 1 at final EOF; for example, `m4wrap(cleanup())`.
`errprint`	Prints its argument on the diagnostic output file.
`dumpdef`	Prints current names and definitions, either for the named items or for all, if no arguments are specified.
`traceon`	Without arguments, turns on tracing for all macros (including built-in macros); 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)

NAME

machid – Gives truth value about processor type

SYNOPSIS

```
crayxmp
crayymp
crayympe
crayympel
crayc90
crayj90
crayts
crayt3d
crayt3e
pdp11
sparc
sun
u370
u3b
u3b5
vax
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The machid utility gives the truth value about the processor type.

EXIT STATUS

The following will return a true value (exit code of 0) if you are on a processor that the command name indicates:

crayc90	True if you are on a CRAY C90 system.
crayj90	True if you are on a CRAY J90 system.
crayts	True if you are on a CRAY T90 system.
crayct3d	True if you are on a CRAY T3D system.
crayct3e	True if you are on a CRAY T3E system.
crayxmp	True if you are on a CRAY Y-MP or a CRAY X-MP system.
crayymp	True if you are on a CRAY Y-MP system.
crayympe	True if you are on a system running IOS model E.
crayympel	True if you are on a CRAY Y-MP EL system.
pdp11	True if you are on a PDP-11/45 or PDP-11/70, always false.

`sparc`	True if you are on a computer using a SPARC-family processor, always false.
`sun`	True if you are on a Sun system, always false.
`u370`	True if you are on a UNIX/370 system, always false.
`u3b5`	True if you are on a 3B5 system, always false.
`u3b`	True if you are on a 3B20S, always false.
`vax`	True if you are on a VAX-11/750 or VAX-11/780, always false.

The commands that do not apply will return a false (nonzero) value. These commands are often used within `make`(1) makefiles and shell procedures to increase portability.

SEE ALSO

`make`(1), `sh`(1), `target`(1), `test`(1), `true`(1)

NAME

`mail` – Invokes an electronic message system

SYNOPSIS

```
mail [-e] [-f file]
mail [-e -p] [-q] [-r] [-f file]
mail [-t] persons
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The `mail` utility invokes an electronic mail system. It can be used to both send and receive mail; you can also use it, for example, from a script.

Receiving Mail

When you log in, the system notifies you if you have mail. It also tells you if new mail arrives while you are using `mail`.

The following options control the way in which received mail is read:

`-e` Prevents mail from being displayed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

`-f` *file* Reads `mail` from *file* (such as `mbox`), rather than the default mail file (`/usr/mail/$LOGNAME`).

`-p` Displays all mail without prompting for disposition.

`-q` Terminates `mail` after interrupts. Usually, an interrupt causes the termination of only the message being displayed.

`-r` Displays messages in first-in, first-out order.

The `mail` command without arguments prints your mail, message by message, in last-in, first-out order. For each message, you are prompted with a `?`, and your response determines the disposition of the message. `mail` also recognizes the following commands when you are reading received mail:

`<newline>` Goes on to next message or stops if there are no more messages.

`+` Functions the same as a newline character.

`d` Deletes the message and goes on to the next message; it deletes only messages at your active security label.

`p`	Prints (displays) message again. It saves a message at your active security label, which may cause the message to be relabeled.
`-`	Returns to previous message, even if it was deleted.
`s` [*files*]	Saves the message, including its header, in the specified *files* (`mbox` is default). If the file to which the message is being saved already exists, it appends the message to it. It saves a message at your active security label, which may cause the message to be relabeled.
`w` [*files*]	Saves the message, without its header (the line that contains the sender's name and postmark), in the specified *files* (`mbox` is default). It saves a message at your active security label, which may cause the message to be relabeled.
`m` [*persons*]	Mails the message to the specified *persons* (default is to you). The message is labeled at your active security label when it is mailed.
`q`	Puts undeleted mail back in the mail file (`/usr/mail/$LOGNAME`) and stops. Mail messages that are not at your security label are not modified.
`<CONTROL-d>`	Functions the same as `q`.
`x`	Puts all mail, including deleted mail, back unchanged in the mail file (`/usr/mail/$LOGNAME`) and stops.
`!`*command*	Escapes to the shell to execute *command*.
`*`	Prints a command summary.

The following options control the way in which received mail is read:

`-e`	Prevents mail from being displayed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.
`-f` *file*	Reads `mail` from *file* (such as `mbox`), rather than the default mail file (`/usr/mail/$LOGNAME`).
`-p`	Displays all mail without prompting for disposition.
`-q`	Terminates `mail` after interrupts. Usually, an interrupt causes the termination of only the message being displayed.
`-r`	Displays messages in first-in, first-out order.

Sending Mail

The persons to whom you send mail (*persons*) are usually user names recognized by `login`(1). If a person being sent mail is not recognized, or if `mail` is interrupted during input, the mail is saved in the `$HOME/dead.letter` file, which can be edited and resent. The dead letter file is overwritten each time mail is mis-sent.

The following option and argument control the way in which mail is sent:

`-t`	Includes all *persons* to whom mail was sent in a line in each recipient's mail header.

persons When you specify *persons*, `mail` takes the standard input up to an end-of-file (or up to a line that consists of just a `.`) and adds it to each person's mail file (`/usr/mail/$LOGNAME`). The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message (that is, `From . . .`) are preceded with a `>`.

You can manipulate the mail file (`/usr/mail/$LOGNAME`) 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
```

This causes all mail sent to the owner of the mail file to be forwarded to *person*. This is especially useful for forwarding all of a person's mail to one machine in a multiple-machine environment. For forwarding to work properly, the mail file should have `mail` as group ID and the group permission should be read-write.

NOTES

If your system is using multilevel directories (MLDs), you must define the directories that contain your system mailbox and your user saved mailbox as a MLD. Your security administrator should be responsible for defining MLDs.

If your system mailbox is not set up as an MLD, mail delivery to users at more than one security label will be disrupted.

If your saved mailbox directory is not defined as an MLD, you will be able to save mail messages only when you are logged in at the security label of your saved mailbox directory.

Depending on system configuration, you may not be allowed to see announcements of received mail at labels to which you do not have mandatory access control (MAC) read access. If you are allowed to receive announcements and you have mail at a label to which you do not have MAC read access, you will be informed that you have unreadable mail at a specific label.

If mail at a certain label is forwarded to another system, you will receive notification in the following form:

```
Your mail at <label> is being forwarded to <destination>.
```

This notification is given for mail at labels to which you have MAC read access.

Although `mail` allows you to save any mail message you receive, the saved version of the mail message is created at the label at which it was read, not necessarily the label at which it was sent.

Although you can forward or send any mail message, the message is always transmitted at your active label. If you decided to forward a message that has a lower security label than your active security label, the message is relabeled with your active security label and then forwarded.

The `mail` command uses `sendmail`(8) as the mechanism to send mail to a remote system.

EXIT STATUS

The `mail` utility exits with the following values:

0 Successful completion when the user had mail.

1 The user had no mail or an initialization error.

>1 An error occurred after initialization.

BUGS

Race conditions sometimes result in a failure to remove a lock file.

After an interrupt, the next message may not be printed; to force printing type a `p`.

A user's mailbox is labeled with the security label of the mail that is contained in the mailbox. Any attempt to send mail to a user at a different security label will fail, and mail can be lost.

EXAMPLES

The following command sends the mail message shown on the second line to users `joe` and `sam`:

```
mail joe sam
This should send mail to Joe and Sam.
.
```

The following command sends the file `memo` to user `sue`:

```
mail sue <memo
```

FILES

`/etc/udb`	User validation file that contains user control limits
`HOME/dead.letter`	Text that could not be mailed
`HOME/mbox`	Saved mail
`/tmp/ma*`	Temporary file
`/usr/mail/*.lock`	Lock for mail directory
`/usr/mail/`*user*	Incoming mail for *user* (the mail file)

SEE ALSO

`login`(1), `write`(1)

`chown`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

`sendmail`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

`mailq` – Prints the contents of the mail queue

SYNOPSIS

`mailq [-v]`

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `mailq` utility prints a summary of the mail messages queued for future delivery.

The first line printed for each message includes its internal identifier used on the host, its size (in bytes), the date and time it was accepted into the queue, and its envelope sender. The second line includes the error that caused the message to be retained in the queue; an error message will not be present if the message is being processed for the first time.

The `mailq` utility is identical to `sendmail -bp`.

The `mailq` command accepts the following flag:

`-v` Prints verbose information. The first line printed for each message will also include the message priority and an indicator ("+") if a warning message has been sent. Additional lines may be present, indicating the "controlling user." These indicate the owner of any programs that are executed on behalf of a message and the alias name (if any) that expanded the command.

The `mailq` utility exits with a value of 0, if successful, and >0 when an error occurs.

NOTES

If this utility is installed with a privilege assignment list (PAL), a user who is assigned the following privilege text upon execution of this command is allowed to perform the actions shown:

Privilege Text	Action
`daemon`	Allowed to see all information in the mail queue
`mailq`	Allowed to see all information in the mail queue

If this utility is installed with a PAL, a user with one of the following active categories is allowed to perform the action shown:

Active Category	Action
`system`, `secadm`	Allowed to see all information in the mail queue

If the PRIV_SU configuration option is enabled, the super user is allowed to see all information in the mail queue.

SEE ALSO

sendmail(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

`mailx` – Invokes an electronic message processing system

SYNOPSIS

Send Mode:

`mailx` [`-h` *number*] [`-r` *address*] [`-s` *subject*] [`-F`] *user...*

Receive Mode:

```
mailx -e
mailx [-d] [-H] [-i] [-n] [-N] [-T file] [-u user] [-U] [-V]
mailx -f [-d] [-H] [-i] [-I] [-n] [-N] [-T file] [-U] [-V] [file]
```

Obsolescent version; may not be supported in future releases:

`mailx` [`-f` [*file*]] [`-d`] [`-H`] [`-i`] [`-I`] [`-n`] [`-N`] [`-T` *file*] [`-U`] [`-V`]

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4
AT&T extensions (`-d`, `-I`, `-T`, `-U`, and `-V` options)

STANDARDS

POSIX, XPG4

DESCRIPTION

The `mailx` utility invokes an electronic mail system. You can use it to send and receive mail.

When you are reading mail, `mailx` lets you save, delete, and respond to messages. When you are sending mail, `mailx` lets you edit, review, and perform other modifications to the messages as you enter them.

The `mailx` utility stores incoming mail in a standard file (*mailbox*) for each user. When you call `mailx` to read messages, the *mailbox* is the default place to find them. As `mailx` reads messages, it marks them to be moved to a secondary file for storage, unless you specify that you want something else done with them. This secondary file is called the `mbox`, and it usually is located in the user's `HOME` directory (see `MBOX` in the ENVIRONMENT VARIABLES section for a description of this file). Messages remain in this file until you remove them.

You can access a secondary file by using the `-f` option of the `mailx` utility. Messages in the secondary file can then be read or otherwise processed using the same `mailx` commands as in the primary *mailbox*.

The operands that follow options are assumed to be destinations (or recipients). If you do not specify recipients, `mailx` attempts to read messages from the system *mailbox*.

The `mailx` utility accepts the following options:

`-d`	Turns on debugging output. This option is not recommended.
`-e`	Tests for presence of mail. If mail to read exists, `mailx` prints nothing and exits with a successful return code.
`-f` [*file*]	Reads messages from *file*, rather than *mailbox*. If you omit *file*, `mbox` is used.
`-F`	Records the message in a file named after the first recipient. If set (see the ENVIRONMENT VARIABLES section), this option overrides the `record` variable.
`-h` *number*	Specifies the number of network "hops" made so far. This option is provided for network software to avoid infinite delivery loops.
`-H`	Prints only header summary.
`-i`	Ignores interrupts. See also `ignore` in the Internal Variables subsection.
`-I`	Includes the newsgroup and article-ID header lines when printing mail messages. You must specify the `-f` option with this option.
`-n`	Prevents initialization from the system default `mailx.rc` file.
`-N`	Prevents printing of initial header summary.
`-r` *address*	Passes *address* to network delivery software.
`-s` *subject*	Sets the subject header field to *subject*.
`-T` *file*	Records message-ID and article-ID header lines in *file* after the message is read. This option also sets the `-I` option.
`-u` *user*	Reads the system mailbox that belongs to *user*. This is successful only if the invoking user has the appropriate privileges to read the system mailbox of that user.
`-U`	Converts uucp style addresses to Internet standards. Overrides the `conv` internal variable. Disables all tilde commands. This option is effective only if the system mailbox that belongs to *user* is not read protected.
`-V`	Prints the `mailx` version number and exits.
user...	Recipients of mail.
file	Read messages from *file*, rather than *mailbox*. If you omit *file*, `mbox` is used.

When mail is being read, `mailx` is in command mode. A header summary of the first several messages is displayed, followed by a prompt that indicates that `mailx` can accept regular commands (see the Commands subsection). When mail is being sent, `mailx` is in *input mode*. If you omit a subject on the command line, a prompt for the subject is printed. As you type the message, `mailx` reads the message and stores it in a temporary file. You can enter commands by beginning a line with the tilde (`~`) escape character, followed by one command letter and optional arguments. See the Tilde Escapes subsection for a summary of these commands.

At any time, `mailx`'s behavior is governed by a set of internal variables. These are flags and valued parameters that are set and cleared by using the `set` and `unset` commands. See the Internal Variables subsection for a summary of these parameters.

Recipients listed on the command line can be of three types: login names, shell commands, or alias groups. Login names can be any network address, including mixed-network addressing. If the recipient name begins with a pipe (`|`) symbol, the rest of the name is considered a shell command through which to pipe the message. This provides an automatic interface with any program that reads the standard input, such as `lp`(1), for recording on paper outgoing mail. Alias groups, set by the alias (`a`) command (see the Commands subsection) are lists of recipients of any type.

Regular commands are of the following form:

[*command*] [*msglist*] [*arguments*]

If you do not specify a command in command mode, the print command (`p`) is assumed. In input mode, commands are recognized by the escape character, and lines not treated as commands are considered input for the message.

Each message is assigned a sequential number, and when it is a current message, it is marked by a `>` in the header summary. Many commands take an optional list of messages (*msglist*) on which to operate, which defaults to the current message. A *msglist* is a list of message specifications separated by spaces that can include the following:

n	Message number *n*
`.`	Current message
`^`	First undeleted message
`$`	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*; *c* is one of the following:

- `d` Deleted messages
- `n` New messages
- `o` Old messages
- `r` Read messages
- `u` Unread messages

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, when expected, are expanded using the typical shell conventions (see `sh`(1)). Special characters are recognized by certain commands, and are documented with the commands that follow.

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: `!`, `Copy`, `edit`, `followup`, `Followup`, `hold`, `mail`, `preserve`, `reply`, `Reply`, `shell`, and `visual`. Any errors in the start-up file cause the remaining lines in the file to be ignored. The `.mailrc` file is optional, and it must be constructed locally.

Commands

The following is a complete list of `mailx` commands:

`!`*command* Invokes the command interpreter specified by `SHELL`. If you set the bang (`!`) variable, each unescaped occurrence of `!` in *command* is replaced with the command executed by the previous `!` command or `~!` tilde escape. See `SHELL` in the ENVIRONMENT VARIABLES section.

 # comment Null command (comment). This might be useful in `.mailrc` files.

`=` Prints the current message number.

`?` Prints a summary of commands.

`a`[`lias`] *alias names* ...
`g`[`roup`] *alias names* ...
Declares an alias for the given names. When *alias* is used as a recipient, the *names* will be substituted. Useful in the `.mailrc` file.

`alt`[`ernates`] *names* ...
Declares a list of alternative names for your login. When responding to a message, these names are removed from the list of recipients for the response. Without arguments, `alternates` prints the current list of alternative names. See also `allnet` in the Internal Variables subsection.

`cd` [*directory*]
`ch`[`dir`] [*directory*] Changes directory. If you omit *directory*, `$HOME` is used.

c[opy] [*file*]
c[opy] [*msglist*] *file* Copies messages to the file without marking the messages as saved; otherwise, it is equivalent to the `save` command. If a message is labeled at a lower security label, the copied message is relabeled with your active security label.

C[opy] [*msglist*] Saves 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, it is equivalent to the `save` command. If a message is labeled at a lower security label, the copied message is relabeled with your active security label.

d[elete] [*msglist*] Deletes messages from the *mailbox*. If you set `autoprint`, the next message after the last one deleted is printed (see the Internal Variables subsection). It deletes messages only at your active security label.

di[scard] [*header-field* ...]
ig[nore] [*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 `Print` and `Type` commands override this command.

dp [*msglist*]
dt [*msglist*] Deletes the specified messages from the *mailbox* and prints the next message after the last one deleted. Roughly equivalent to a `delete` command, followed by a `print` command. Deletes messages only at your active security label.

ec[ho] *string* ... Echoes the given strings (such as `echo`(1)).

e[dit] [*msglist*] Edits the given messages. The messages are placed in a temporary file, and the `EDITOR` environment variable is used to get the name of the editor (see the ENVIRONMENT VARIABLES section). Default editor is `ed`(1). Editing of a message is done at your current security label. If the message cannot be written back to the mail file at that label, the edited message is discarded when the mail box is closed.

ex[it]
x[it] Exits from `mailx`, without changing the *mailbox*. No messages are saved in the `mbox` (see also `quit`).

fi[le] [*file*]
fold[er] [*file*] Quits from the current file of messages and reads 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*

+*file* The named file in the `folder` directory (see the `folder` variable)

The default file is the current *mailbox*.

`folders` Prints the names of the files in the directory set by the `folder` variable (see the Internal Variables subsection).

`fo`[`llowup`] [*message*]

Responds 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 `Followup`, `Save`, and `Copy` commands and `outfolder` in the Internal Variables subsection.

`F`[`ollowup`] [*msglist*] Responds 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 `followup`, `Save`, and `Copy` commands, and `outfolder` in the Internal Variables subsection.

`f`[`rom`] [*msglist*] Prints the header summary for the specified messages.

`g`[`roup`] *alias names* . . .
`a`[`lias`] *alias names* . . .

Declares an alias for the given *names*. When *alias* is used as a recipient, the *names* will be substituted. Useful in the `.mailrc` file.

`h`[`eaders`] [*message*] Prints the page of headers that includes the specified message. The `screen` variable sets the number of headers per page (see the Internal Variables subsection). See also the `z` command.

`hel`[`p`] Prints a summary of commands.

`ho`[`ld`] [*msglist*]
`pre`[`serve`] [*msglist*] Holds the specified messages in the *mailbox*.

`i`[`f`] *s*|*r*
`""`*mail-command*s
`el`[`se`]
`""`*mail-command*s
`en`[`dif`] Specifies conditional execution; *s* executes the following *mail-command*s, up to an `else` or `endif`, if the program is in send mode, and *r* executes the *mail-command*s only in receive mode. Useful in the `.mailrc` file.

`ig`[`nore`] *header-field* . . .
`di`[`scard`] *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`. When the message is saved, all fields are included. The `Print` and `Type` commands override this command.

`l`[`ist`]	Prints all commands available. No explanation is given.
`m`[`ail`] *name* ...	Mails a message to the specified users. It labels the mailed message at your active security label.
`M`[`ail`] *name*	Mails a message to the specified user and records a copy of it in a file with the same name as the user.
`mb`[`ox`] [*msglist*]	Arranges for the specified messages to be in the standard `mbox` save file when `mailx` terminates normally. See `MBOX` in the ENVIRONMENT VARIABLES section for a description of this file. See also the `exit` and `quit` commands.
`n`[`ext`] [*message*]	Goes to next message that matches *message*. A *msglist* can be specified; however, the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, because the name would be taken as a command in the absence of a real command. See the previous discussion of *msglist*s for a description of possible message specifications.
`pi`[`pe`] [[*msglist*] *command*]	
`|` [[*msglist*] *command*]	Pipes the message through the given *command*. The message is treated as if it were read. Without arguments, the current message is piped through the command specified by the value of the `cmd` variable. If the `page` variable is set, it inserts a `<form-feed>` character after each message (see the Internal Variables subsection).
`pre`[`serve`] [*msglist*]	
`ho`[`ld`] [*msglist*]	Preserves the specified messages in the *mailbox*.
`P`[`rint`] [*msglist*]	
`T`[`ype`] [*msglist*]	Prints the specified messages on the screen, including all header fields. Overrides suppression of fields by the `ignore` command.
`p`[`rint`] [*msglist*]	
`t`[`ype`] [*msglist*]	Prints the specified messages. If you set `crt`, messages longer than the number of lines specified by the `crt` variable are paged through the command specified by the `PAGER` environment variable. The default command is `pg`(1) in the ENVIRONMENT VARIABLES section.
`q`[`uit`]	Exits from `mailx`, storing messages that were read in `mbox` and unread messages in the *mailbox*. Deletes messages that have been explicitly saved in a file. Messages that are saved in `mbox` are relabeled, if necessary, at your active security label.
`R`[`eply`] [*msglist*]	
`R`[`espond`] [*msglist*]	Sends a response to the author of each message in the *msglist*. The subject line is taken from the first message. If you set `record` to a file name, the response is saved at the end of that file (see the Internal Variables subsection). Any response to a message is labeled at your active security label, even if the reply is to a message at a lower security label.

r[eply] [*message*]
r[espond] [*message*] Replies to the specified message, including all other recipients of the message. If you set `record` to a file name, the response is saved at the end of that file (see the Internal Variables subsection).

S[ave] [*msglist*] Saves 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 `Copy`, `followup`, and `Followup` commands, and `outfolder` in the Internal Variables subsection. Messages can be saved only at your active security label; if a message with a lower security label is saved, it is relabeled with your active security label.

s[ave] [*file*]
s[ave] [*msglist*] *file* Saves the specified messages in the given file. The file is created if it does not exist, or the message is appended to it if it does exist. The message is deleted from the *mailbox* when `mailx` terminates, unless `keepsave` is set (see in the Internal Variables subsection and the `exit` and `quit` commands). Messages can be saved only at your active security label; if a message with a lower security label is saved, it is relabeled with your active security label.

se[t]
se[t] *name*
se[t] *name=string*
se[t] *name=number* Defines a variable called *name*. The variable can be given a null, string, or numeric value. `set` by itself prints all defined variables and their values. See the Internal Variables subsection for detailed descriptions of the `mailx` variables.

sh[ell] Invokes an interactive shell (see `SHELL` in the ENVIRONMENT VARIABLES section).

si[ze] [*msglist*] Prints the size of the specified messages in number of characters

so[urce] *file* Reads commands from the specified file and returns to command mode.

to[p] [*msglist*] Prints the top few lines of the specified messages. If you set the `toplines` variable, it is taken as the number of lines to print (see in the Internal Variables subsection). The default is 5.

tou[ch] [*msglist*] Touches the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the `mbox` on normal termination. See `exit` and `quit`.

T[ype] [*msglist*]
P[rint] [*msglist*] Prints the specified messages on the screen, including all header fields. Overrides suppression of fields by the `ignore` command.

t[ype] [*msglist*]

p[rint] [*msglist*] Prints the specified messages. If you set `crt`, messages that are longer than the number of lines specified by the `crt` variable are paged through the command specified by the `PAGER` environment variable. The default command is `pg`(1) in the ENVIRONMENT VARIABLES section.

u[ndelete] [*msglist*] Restores the specified deleted messages. Will restore only messages that were deleted in the current mail session. If you set `autoprint`, the last message of those restored is printed (see the Internal Variables subsection).

undi[scard]
unig[nore] Removes the specified header fields from the list being ignored.

uns[et] *name* ... Erases the specified variables. If the variable was imported from the execution environment (that is, a shell variable), it cannot be erased.

ve[rsion] Prints the current version and release date.

v[isual] [*msglist*] Edits the specified messages by using a screen editor. The messages are placed in a temporary file and the `VISUAL` environment variable is used to get the name of the editor (see the ENVIRONMENT VARIABLES section). Editing of a message is done at your current security label. If the message cannot be written back to the mail file at that label, the edited message is discarded when the mail box is closed.

w[rite] [*msglist*] *file* Writes the given messages on the specified file, minus the header and trailing blank line; otherwise, it is equivalent to the `save` command. Messages can be saved only at your active security label. If a message with a lower security label is saved, it is relabeled with your active security label.

x[it]
ex[it] Exits from `mailx` without changing the *mailbox*. No messages are saved in the `mbox` (see also `quit`).

z[+|-] Scrolls the header display forward or backward one screenful. The `screen` variable sets the number of headers displayed (see the Internal Variables subsection).

Tilde Escapes

You can enter the following commands only from input mode, by beginning a line with the tilde escape character (~). See `escape` (see the Internal Variables subsection) for changing this special character.

~! *command* Escapes to the shell.

~. Simulates end of file (terminates message input). **Warning:** Users using `rlogin` to connect to Cray Research systems should not use the tilde-dot (~.) character sequence to simulate end-of-file. The character sequence tilde-dot (~.) will be interpreted by `rlogin` first to disconnect the user from the Cray Research system. Use `<CTRL-d>` to terminate message input.

~: *mail-command*
~_ *mail-command* Performs the command-level request. Valid only when sending a message while reading mail.

~?	Prints a summary of tilde escapes.
~A	Inserts the autograph string.
~a	Inserts the autograph string `sign` into the message (see the Internal Variables subsection).
~b *name* ...	Adds the *name* to the blind carbon copy (`Bcc`) list.
~c *name* ...	Adds the *name* to the carbon copy (`Cc`) list.
~d	Reads in the `dead.letter` file. See `DEAD` in the Internal Variables subsection for a description of this file.
~e	Invokes the editor on the partial message. See also `EDITOR` in the ENVIRONMENT VARIABLES section.
~f [*msglist*]	Forwards the specified messages. Inserts the messages into the message, without alteration.
~h	Prompts for `Subject line` and `To`, `Cc`, and `Bcc` lists. If the field is displayed with an initial value, you can edit it as if you had just typed it.
~i *string*	Inserts the value of the specified variable into the text of the message (for example, ~A is equivalent to `~i Sign`).
~m [*msglist*]	Inserts 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	Prints the message being entered.
~q	Quits 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` in the ENVIRONMENT VARIABLES section for a description of this file.
~r *file* ~< *file* ~< !*command*	Reads 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* ...	Sets the subject line to *string*.
~t *name* ...	Adds the given *name* to the list.
~v	Invokes a preferred screen editor on the partial message. See also the `VISUAL` environment variable in the ENVIRONMENT VARIABLES section.
~w *file*	Writes the partial message onto the specified file, without the header.
~x	Exits as with ~q, except the message is not saved in `dead.letter`.
~| *command*	Pipes the body of the message through the specified *command*. If the *command* returns a successful exit status, the output of the command replaces the message.

Internal Variables

The following variables are internal `mailx` variables. You can set each internal variable by using the `mailx set` command at any time. To erase variables use the `unset` and `set no`*name* commands.

`allnet` — Treats all network names whose last component (login name) match as identical. This causes the *msglist* message specifications to behave similarly. Default is `noallnet`. See also the `alternates` command and the `metoo` variable.

`append` — On termination, appends messages to the end of the `mbox` file instead of prepending them. Default is `noappend`.

`ask`
`asksub` — Prompts for the subject if it is not specified on the command line by using the `-s` option. Enabled by default.

`askbcc` — Prompts for the `Bcc` list after the subject is entered. Default is `noaskcc`.

`askcc` — Prompts for the `Cc` list after the subject is entered. Default is `noaskcc`.

`autoprint` — Enables automatic printing of messages after `delete` and `undelete` commands. Default is `noautoprint`.

`bang` — Enables the special-casing of exclamation points (`!`) in shell escape command lines as in `vi`(1). Default is `nobang`.

`cmd`=*command* — Sets the default command for the `pipe` command. No default value.

`conv`=*conversion* — Converts `uucp` addresses to the specified address style. The only valid conversion is *internet*, which requires a mail delivery program that conforms to the RFC 822 standard for electronic mail addressing. By default, conversion is disabled. See also `sendmail`(8) and the `-U` command-line option.

`crt`=*number* — Pipes messages that have more than *number* lines through the command specified by the value of the `PAGER` environment variable (`pg`(1)). Disabled by default.

`debug` — Enables verbose diagnostics for debugging. Messages are not delivered. Default is `nodebug`.

`dot` — Reads a period on a line by itself during input from a terminal as end-of-file. Default is `nodot`.

`escape`=*c* — Substitutes *c* for the ~ escape character.

`flipr` — Reverses the meanings of the `R` and `r` commands. The default is `noflipr`.

`folder`=*directory* — Saves standard mail files. Expands user-specified file names that begin with a plus (`+`) 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. To use the plus (`+`) construct on a `mailx` command line, `folder` must be an exported `sh` environment variable. No default exists for the `folder` variable. See also `outfolder` in this subsection.

`header` Enables printing of the header summary when entering `mailx`. Enabled by default.

`hold` Preserves all messages that are read in the *mailbox* instead of putting them in the standard `mbox` save file. Default is `nohold`.

`ignore` Ignores interrupts while entering messages. Handy for noisy dial-up lines. Default is `noignore`.

`ignoreeof` Ignores 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 the `dot` variable in this subsection.

`indentprefix`=*string*
A string to be prefixed to each line that is inserted into the message by the `~m` command escape. The default for this variable is one tab character.

`keep` When the *mailbox* is empty, truncates it to a length of 0 instead of removing it. Disabled by default.

`keepsave` Keeps messages that have been saved in other files in the *mailbox* instead of deleting them. Default is `nokeepsave`.

`metoo` If your login appears as a recipient, prevents its deletion from the list. Default is `nometoo`.

`onehop` When responding to a message that was originally sent to several recipients, the other recipient addresses are usually 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 (that is, one hop away).

`outfolder` Locates the files used to record outgoing messages in the directory specified by the `folder` variable, unless the path name is absolute. Default is `nooutfolder`. See `folder` in this subsection and the `Save`, `Copy`, `followup`, and `Followup` commands.

`page` Inserts a `<form-feed>` after each message sent through the pipe, when used with the pipe command. Default is `nopage`.

`prompt`=*string* Sets the *command mode* prompt to *string*. Default is `?`.

`quiet` Refrains from printing the opening message and version when entering `mailx`. Default is `noquiet`.

`record`=*file* Records all outgoing mail in *file*. Disabled by default. See also `outfolder` in this subsection.

`save` Enables saving of messages in `dead.letter` on interrupt or delivery error. See the `DEAD` environment variable in the ENVIRONMENT VARIABLES section for a description of this file. Enabled by default.

`screen`=*number* Sets the number of lines in a screenful of headers for the `headers` command.

`sendmail`=*command*	Alternate command for delivering messages. Default is `/usr/lib/sendmail`.
`sendwait`	Waits for background mailer to finish before returning. Default is `nosendwait`.
`showto`	When displaying the header summary and the message is from you, prints the recipient's name instead of the author's name.
`sign`=*string*	Identifies the variable inserted into the text of a message when the `~a` (autograph) command is given. No default (see also `~i` in the Tilde Escapes subsection).
`Sign`=*string*	Identifies the variable inserted into the text of a message when the `~A` command is specified. No default (see also `~i` in the Tilde Escapes subsection).
`toplines`=*number*	Indicates the number of lines of header to print with the `top` command. Default is 5.

NOTES

If your saved mail box directory is not defined as a multilevel directory (MLD), you can save mail messages only when logged in at the security label of your saved mail box directory.

Depending on system configuration, you may not be allowed to see announcements of received mail at labels to which you do not have MAC read access. If you are allowed to receive announcements and you have mail at a label to which you do not have MAC read access, you will be informed that you have unreadable mail at a specific label.

If mail at a certain label is forwarded to another system, you will receive notification in the following form:

```
Your mail at <label> is being forwarded to <destination>.
```

This notification is given for mail at labels to which you have MAC read access.

Although `mailx` allows you to save any mail message you receive, the saved version of the mail message is created at the label at which it was read, not necessarily the label at which it was sent.

Although you can forward or send any mail message, the message is transmitted at your active label. If you decide to forward a message that has a lower label than your active label, the message is relabeled with your active label and then forwarded.

ENVIRONMENT VARIABLES

Following are the environment variables for `mailx`:

`DEAD`	The path name of the file in which to save partial letters if untimely interrupts or delivery errors. Default is `$HOME/dead.letter`.
`EDITOR`	The command to run when the `edit` or `~e` command is used. Default is `ed`(1).
`HOME`	The user's base of operations; the user's home directory.
`LISTER`	A string that represents the command for writing the contents of the `folder` directory to standard output when the `folders` command is specified. The default is `ls`(1).

MAILRC The path name of the start-up file. Default is $HOME/.mailrc.

MBOX The path name of the file to save messages that have been read. The exit command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

PAGER A string that represents an output filtering and/or pagination command for writing the output to the terminal. When standard output is a terminal device, the message output is piped through the command if the mailx internal variable crt is set to a value less than the number of lines in the message. Default is pg(1).

SHELL Identifies the name of a preferred command interpreter. Default is sh(1).

TERM If the internal variable screen is not specified, identifies the name of the terminal type to determine the number of lines in a screenful of headers.

VISUAL The name of a preferred screen editor. When the visual command or ~v command-escape is used, this editor will be invoked. Default is vi(1).

EXIT STATUS

The -e option is specified, mailx exits with one of the following values:

0 Mail was found.

>0 Mail was not found or an error occurred.

Otherwise, mailx exits with one of the following values:

0 Successful completion.

>0 An error occurred.

BUGS

You can use the -h and -r options only if mailx is using a delivery system program other than /usr/bin/rmail.

When *command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

You cannot unset Internal variables imported from the execution environment.

The mailx utility does not fully support the full Internet addressing.

The sendmail(8) utility (the default mail delivery program) interprets a message that is a line consisting only of a ‘‘.’’ as the end of the message.

FILES

$HOME/.mailrc	Personal start-up file
$HOME/mbox	Secondary storage file
/tmp/R[emqsx]*	Temporary files

`/usr/lib/mailx/mailx.help*`	Help message files
`/usr/lib/mailx/mailx.rc`	Global start-up file
`/usr/mail/*`	Post office directory

SEE ALSO

`ls`(1), `mail`(1), `pg`(1), `sh`(1)

`sendmail`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

make – Maintains, updates, and regenerates groups of programs

SYNOPSIS

make [-k] [-d] [-e] [-f *makefile*]... [-i] [-l] [-n] [-o] [-p] [-q] [-r] [-s] [-t] [*macro=name*]... [*target_name...*]

make [-S] [-d] [-e] [-f *makefile*]... [-i] [-l] [-n] [-o] [-p] [-q] [-r] [-s] [-t] [*macro=name*]... [*target_name...*]

IMPLEMENTATION

Cray PVP systems

STANDARDS

POSIX, XPG4
AT&T extensions (-d option)

DESCRIPTION

The make utility executes commands in *makefile* to update one or more target *target_name*s, which are typically programs.

The make utility examines time relationships and updates those derived files (called targets) that have modified times earlier than the modified times of the files (called prerequisites) from which they are derived. A description file (makefile) contains a description of the relationships between files, and the commands that must be executed to update the targets to reflect changes in their prerequisites. Each specification, or rule, consists of a target, optional prerequisites, and optional commands to be executed when a prerequisite is newer than the target. There are two types of rules:

- Inference rules, which have one target name with at least one period (.) and no slash (/)
- Target rules, which can have more than one target name

In addition, make has a collection of built-in macros and inference rules that infer prerequisite relationships to simplify maintenance of programs.

The system administrator may define a file named /etc/MAKEFILE, which can contain an additional set of macros, special targets, and inference rules that modify the built-in rules or add more definitions to them. This file enables a site to define a specific make environment valid for the entire site. The /etc/MAKEFILE file is automatically read in when make is invoked and before any user makefile is processed, unless the -l option is used.

When the -f option is not present, makefile, Makefile, s.makefile, and s.Makefile are tried in order. If *makefile* is -, standard input will be taken. More than one -f *makefile* argument pair may appear.

The make utility updates a target only if it depends on files that are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are considered out-of-date.

After make has ensured that all of the prerequisites of a target are up-to-date, and if the target is out-of-date, the commands associated with the target entry are executed. If there are no commands listed for the target, the target is treated as up-to-date.

The following options are supported:

-d Writes to standard error detailed information on files and times examined.

-e Causes environment variables to override macro assignments within makefiles.

-f *makefile* Specifies a different makefile. The argument *makefile* is a pathname of a description file, also referred to as the *makefile*. A file name of - denotes standard input. There can be multiple occurrences of this option, and they are processed in the order specified. The contents of *makefile* override the built-in rules if they are present.

-i Ignores error codes returned by invoked commands. This mode is the same as if the special target name .IGNORE appears in the description file.

-k Continues to update other targets that do not depend on the current target if a non-ignored error occurs while executing the commands to bring a target up-to-date.

-l Ignores the file /etc/MAKEFILE and does not read it in when make is invoked.

-n Writes commands to standard output, but does not execute them. Lines with a plus-sign (+) prefix are executed. Lines with an at-sign (@) prefix are written to standard output.

-o Prints the contents of the /etc/MAKEFILE file (if it exists).

-p Writes to standard output the complete set of macro definitions and target descriptions.

-q Returns a zero exit value if the target file is up-to-date; otherwise, returns an exit value of 1. Targets are not updated. A command-line (associated with the targets) with a plus-sign (+) prefix is executed.

-r Clears the suffix list and does not use the built-in rules.

-s Does not write command lines or touch messages (see -t) before executing. This mode is the same as entered if the special target name .SILENT appears in the description file.

-S Terminates make if an error occurs while executing the commands to bring a target up-to-date. This is the default and the opposite of -k.

-t Updates the modification time of each target as though a touch *target* had been executed (see touch(1)). Targets that have prerequisites but no commands, or that are already up-to-date, are not touched in this manner. Write messages to standard output for each target file indicating the name of the file and that it was touched. Normally, the command lines associated with each target are not executed. However, a command line with a plus-sign (+) prefix is executed.

If the -k and -S options are both specified on the command line, by the MAKEFLAGS environment variable or by the MAKEFLAGS macro, the last one evaluated takes precedence. The MAKEFLAGS environment variable is evaluated first and the command line is evaluated second.

The following operands are supported:

macro=name Macro definitions. See the Macros subsection.

target_name Target names. If no target is specified, while make is processing the makefiles, the first target that make encounters that is not a special target or an inference rule is used.

Makefile Syntax

A makefile can contain rules, macro definitions, and comments. There are two kinds of rules: inference rules and target rules. The make utility contains a set of built-in inference rules. If the -r option is present, the built-in rules are not used and the suffix list is cleared. Additional rules of both types can be specified in a makefile. If a rules or macro is defined more than once, the value of the rule or macro is that of the last one specified. Comments start with a # symbol and continue until an unescaped newline character is reached.

The rules in makefiles consist of the following types of lines: target rules (including special targets), inference rules, macro definitions, empty lines, and comments.

When an escaped newline character (one preceded by a \ symbol) is found anywhere in the makefile, it is replaced, along with any leading white space on the following line, with a single space.

Makefile Execution

Command lines can have one or more of the following prefixes: a dash (-), an at sign (@), or a plus sign (+). These modify the way in which make processes the command. When a command is written to standard output, the prefix is not included in the output.

- If the command prefix contains a dash, or the -i option is present, or the special target .IGNORE has either the current target as a prerequisite or has no prerequisites, any error found while executing the command is ignored.

@ If the command prefix contains an at sign and the command-line -n option is not specified, or the -s option is present, or the special target .SILENT has either the current target as a prerequisite or has no prerequisites, the command is not written to standard output before it is executed.

\+ If the command prefix contains a plus sign, this indicates a command line is executed even if -n, -q, or -t is specified.

The -n option specifies printing without execution; however, if the command line contains the string $(MAKE), the line will be executed regardless of the -n option (see the discussion of the MAKEFLAGS macro under the Environment subsection). The -t (touch) option updates the date of target files without executing any commands.

Commands returning nonzero status normally terminate make. The error is ignored when the -i option is present, .IGNORE: appears in the makefile, or the initial character sequence of the command contains a dash. When the -k option is present, work stops on the current entry but continues on other branches that do not depend on that entry.

An interrupt, software termination, hangup, or quit signal received during the execution of a command line causes the associated target to be deleted, unless the target is a prerequisite of the special target `.PRECIOUS`. Text following a semicolon (`;`) character and all following lines that begin with a `<tab>` are shell commands to be executed to update the target.

The first nonempty line that does not begin with a `<tab>` or # begins a new prerequisite or macro definition. Shell commands may be continued across lines with the backslash character followed by a newline character. Everything printed by `make` (except the initial `<tab>`) is passed directly to the shell as is.

Command lines are executed one at a time, each by its own shell. The environment for the command being executed contains all of the variables in the environment of `make`. The macros from the command line to `make` are added to `make`'s environment. If any command-line macro has been defined elsewhere, the command-line value overwrites the existing value. By default, when `make` receives a nonzero exit status from the execution of a command, it terminates with an error message to standard error.

The following example makefile indicates 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
```

Target Rules

Target rules are formatted as follows:

target[*target*...]: [*prerequisite*...][;*command*]
[<tab>*command*
<tab>*command*
...]
(*line that does not begin with* <tab>)

Target entries are specified by a blank-separated, nonnull list of targets, followed by a colon, and then a blank-separated, possibly empty, list of prerequisites. Text following a semicolon, if any, and all following lines that begin with a `<tab>` are command lines to be executed to update the target. The first nonempty line that does not begin with a `<tab>` or # begins a new entry. An empty or blank line, or a line beginning with #, may begin a new entry.

Applications may select target names from the set of characters consisting solely of periods, underscores, digits, and alphabetics.

The following are recognized as special `make` targets:

`.DEFAULT` Uses the commands associated with the name `.DEFAULT` if it exists; these commands are used when a file must be made but there are no explicit commands or relevant built-in rules.

`.IGNORE` Prerequisites of this special target are targets themselves; this causes errors from commands associated with them to be ignored in the same manner as specified by the `-i` option. Subsequent occurrences of `.IGNORE` are added to the list of targets ignoring command errors. If no prerequisites are specified, `make` behaves as if the `-i` option had been specified and errors from all commands associated with all targets are ignored.

`.POSIX` This target is specified without prerequisites or commands. If it appears on a line by itself anywhere in any of the makefiles specified, `make` processes all makefiles in a manner defined by POSIX 1003.2. See the Suffixes subsection.

`.PRECIOUS` Prerequisites of this special target are not removed if `make` receives one of the following signals: quit, hang up, interrupt, and software termination. Subsequent occurrences of `.PRECIOUS` are added to the list of precious files. If no prerequisites are specified, all targets in the makefile are treated as if specified with `.PRECIOUS`.

`.SILENT` Prerequisites of this special target are targets themselves; this causes commands associated with them to not be written to the standard output before they are executed. Subsequent occurrences of `.SILENT` are added to the list of targets with silent commands. If no prerequisites are specified, `make` behaves as if the `-s` option had been specified and no commands or touch messages associated with any target is written to standard output.

`.SUFFIXES` Prerequisites of `.SUFFIXES` are appended to the list of known suffixes and are used in conjunction with the inference rules (see the Inference Rules subsection). If `.SUFFIXES` does not have any prerequisites, the list of known suffixes are cleared. Makefiles do not associate commands with `.SUFFIXES`.

Macros

Entries of the form *string1=string2* are macro definitions. The macro named *string1* is defined as having the value of *string2*, where *string2* is defined as all characters, if any, after the equal sign(`=`), up to a comment character (`#`) or an unescaped newline character. Any blanks immediately before or after the equal sign are ignored.

Subsequent appearances of `$(`*string1*`)` or `$(`*string2*`)` are replaced by *string2*. The parentheses or braces are optional if *string1* is a single character. The macro `$$` is replaced by the single character `$`.

Macro names are made up from the set of characters consisting solely of periods, underscores, digits, and alphabetics from the portable character set (that is, ASCII). A macro name may not contain an equal sign.

Macros can appear anywhere in the makefile. Macros in target lines are evaluated when the target line is read. Macros in command lines are evaluated when the command is executed. Macros in macro definition lines are not evaluated until the new macro being defined is used in a rule or command. A macro that has not been defined evaluates to a null string without causing an error condition.

The forms $(*string1*[:*subst1*=[*subst2*]]) or ${*string1*[:*subst1*=[*subst2*]]} can be used to replace all occurrences of *subst1* with *subst2* when the macro substitution is performed. The *subst1* to be replaced is recognized when it is a suffix at the end of a word in *string1* (a word in this context is defined to be a string delimited by the beginning of the line, a blank, or a newline character).

Macro assignments are accepted from the sources listed below, in the order shown. If a macro name already exists at the time it is being processed, the newer definition replaces the existing definition:

1. Macros defined in make's built-in inference rules.
2. The contents of the environment, including the variables with null values, in the order defined in the environment.
3. Macros defined in the makefile(s), processed in the order specified.
4. Macros specified on the command line.

If the -e option is specified, the order of processing sources 2 and 3 are reversed.

The VPATH macro, consisting of a list of colon-separated directory paths, may be used to specify the location of prerequisite files.

The SHELL macro is treated specially. It is provided by make and set to the path name of the shell command language interpreter, /bin/sh. The SHELL environment variable does not affect the value of the SHELL macro. If SHELL is defined in the makefile or is specified on the command line, it replaces the original value of the SHELL macro, but does not affect the SHELL environment variable.

The MAKEFLAGS environment variable, when processed by make, is assumed to contain any legal input option (except -f, -p, and -r) defined for the command line. Further, upon invocation, make creates 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*, which are master makefiles that call other makefiles (these are used for system builds). The MAKEFLAGS environment variable may also contain option letters without the leading dashes (-) and no separating blanks.

In fact, as noted previously, when the -n option is used, the $(MAKE) command is executed anyway; hence, you can specify 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 makefiles for a software project without actually executing anything.

Inference Rules

Inference rules can be made shorter, as in this example:

```
pgm:    a.o b.o
        cc a.o b.o -o pgm
a.o b.o:        incl.h
```

The shorter form is possible because make has a set of internal rules for building files. A 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 and YFLAGS are used for compiler options to cc(1) and yacc(1), respectively. The previously stated method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule for creating a file with suffix .o from a file with suffix .c is specified as an entry with .c.o: as the target and no prerequisites. 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 prerequisite name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member in the library. Thus, lib(file.o) and $(LIB)(file.o) both refer to an archive library that contains file.o (assuming that 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, in which the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires that *XX* be different from the suffix of the archive member. Thus, lib(file.o) cannot depend on file.o explicitly.

The most common use of the archive interface follows; it assumes that the source files consist of all C-type source code:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
c.a:
        $(CC) -c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.o
```

The .c.a: rule listed previously is built into make and is not necessary 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) $(ARFLAGS) lib $?
        rm $?
        @echo lib is now up-to-date
c.a:;
```

In this example, the substitution mode of the macro expansions is used. The $? list is defined as the set of object file names (inside lib) whose C source files are out-of-date. The substitution mode translates .o to .c. Note also the disabling of the .c.a: rule, which would have created each object file, one by one. This particular construct speeds archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

Internal Macros

The following macros are maintained internally and are useful in writing rules for building targets:

`$*` The `$*` macro represents the file name part of the current target name with the suffix deleted. It is evaluated only for inference rules.

For example, in the `.c.a` inference rule, `$*.o` represents the out-of-date `.o` file that corresponds to the prerequisite `.c` file.

`$@` The `$@` macro represents the full target name of the current target. It is evaluated only for explicitly named dependencies.

For example, in the `.c.a` inference rule, `$@` represents the out-of-date `.a` file to be built. Similarly, in a makefile target rule to build `lib.a` from `file.c`, `$@` represents the out-of-date `lib.a`.

`$<` The `$<` macro is evaluated only for inference rules or the `.DEFAULT` rule. It is the module that is out-of-date with respect to the target (that is, the ''manufactured'' dependent file name). Thus, in the `.c.o` rule, the `$<` macro would evaluate to the `.c` file. Following are two examples for making `.o` files from `.c` files:

```
.c.o:
        cc -c $.c
```

or:

```
.c.o:
        cc -c $<
```

`$?` The `$?` 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, the modules that must be rebuilt).

For example, in a makefile target rule to build `prog` from `file1.o`, `file2.o`, and `file3.o`, and where `prog` is not out of date with respect to `file1.o`, but is out of date with respect to `file2.o` and `file3.o`, `$?` represents `file2.o` and `file3.o`.

`$%` The `$%` macro is evaluated only when the current target is an archive library member of the form *libname*`(`*member*`.o)`. In these cases, `$@` evaluates to *libname* and `$%` evaluates to *member*`.o`. The `$%` macro is evaluated for both target and inference rules.

For example, in a makefile target rule to build `lib.a(file.o)`, `$%` represents `file.o`, as opposed to `$@`, which represents `lib.a`.

Each of the internal macros can have alternative forms. When an uppercase `D` or `F` is appended to any of these macros, the meaning is changed to *directory part* for `D` and *file part* for `F`. The directory part is the path prefix of the file without a trailing slash; for the current directory, the directory part is a period (`.`). When the `$?` macro contains more than one prerequisite file name, the `$(?D)` and `$(?F)` (or `${?D}` and `${?F}`) macros expand to a list of directory name parts and file name parts respectively.

For the target *lib*(*member*.o) and the .*s2*.a rule, the internal macros are defined as follows:

$< *member*.*s2*

$* *member*

$@ *lib*

$? *member*.*s2*

$% *member*.o

Include Files

If the string include or sinclude appears at the beginning of a line in a makefile and is followed by a <blank> or a <tab>, the rest of the line is assumed to be a file name and will be read by the current invocation of make after substituting for any macros. If the file is not readable, it is a fatal error for include, and silently ignored by sinclude. After reading the include file, the remainder of the makefile will be read. Include files can be nested up to 16 levels.

Suffixes

Certain names (for instance, those ending with .o) have prerequisites, such as .c or .s, that can be inferred. If no update commands for such a file appear in *makefile*, and if a prerequisite that can be inferred exists, that prerequisite will be compiled to make the target. In this case, make has inference rules that allow files to be built from other files when the suffixes are processed and the appropriate inference rule is used.

When the special target .POSIX is used, the following rules are defined:

```
.c .c.a .c.o
.f .f.a .f.o
.l.c .l.o
.sh
.y.c .y.o
```

When .POSIX is not used, all of the preceding rules, in addition to the ones following, are defined:

```
.c~.a .c~.c .c~.o .C .C~ .C.o .C~.C .C~.o .C.a .C~.a
.f .f~ .f~.f .f.a .f~.a .f.o .f~.o
.F .F~ .F~.F .F.a .F~.a .F.o .F~.o
.f90 .f90~ .f90~.f90 .f90.a .f90~.a .f90.o .f90~.o
.F90 .F90~ .F90~.F90 .F90.a .F90~.a .F90.o .F90~.o
.h~.h
.l~.c .l~.l .l~.o
.p .p~ .p.a .p~.a .p.o .p~.o .p~.p
.s.a .s~.a .s.o .s~.o .s~.s
.sh~ .sh~.sh
.y~.c .y~.o .y~.y
```

The internal rules for the make utility can be printed in a form suitable for modifications and for inclusion in a user's makefiles. To print the make default built-in rules, use the following standard shell command:

```
make -fpl - 2>/dev/null </dev/null
```

To print the make default built-in rules as modified by the existence of the /etc/MAKEFILE file, use the following standard shell command:

```
make -fp - 2>/dev/null </dev/null
```

Note: If the /etc/MAKEFILE file contains the special target .POSIX, this command will list make default built-in POSIX rules as modified by the contents of /etc/MAKEFILE.

To print only the contents of /etc/MAKEFILE, use the following standard shell command:

```
make -fo - 2>/dev/null </dev/null
```

To print the make built-in POSIX rules, create a file named makefile containing the special target .POSIX and execute the following standard shell command:

```
make -pl 2>/dev/null
```

To print the built-in POSIX rules as modified by the contents of /etc/MAKEFILE, create a file named makefile containing the special target .POSIX and execute the following shell command:

```
make -p 2</dev/null
```

Note: If /etc/MAKEFILE already contains the special target .POSIX, it is not necessary to create the makefile containing this target.

The only peculiarity in this output is the (null) string that printf(3C) prints when handed a null string.

A tilde (~) in the above rules refers to a Source Code Control System (SCCS) file. 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 the make 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 (that is, .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 (such as shell procedures and simple C programs).

Additional suffixes are specified as the prerequisite 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 suffix list when the special target .POSIX is used is as follows:

```
.SUFFIXES: .o .c .y .l .a .sh .f
```

When .POSIX is not used, the list is as follows:

```
.SUFFIXES: .o .c .c~ .C .C~ .f .f~ .F .F~ .f90 .f90~ .F90 .F90~ .p .p~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~
```

The command for printing the internal rules (shown earlier in this subsection) displays the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; `.SUFFIXES:` without prerequisites clears the list of suffixes.

VPATH Macro

When `make` searches for targets and prerequisites, it expects to find them in the current (`.`) directory or in the directory specified by their file names. For example, in the following rule, `make` would look for the file `file.c` in the `.` directory:

```
file.o: file.c
```

The `VPATH` macro allows you to specify alternate paths for the file search. The variable consists of the directory names separated by a colon (`:`), in the style of the `PATH` environmental variable.

The following example specifies a path containing three directories: `newsrc`, `src`, and `obj`.

```
VPATH=newsrc:src:obj
```

The `make` utility will search the directories in the following order: the current (`.`) directory, followed by `newsrc`, `src`, and `obj` for all dependencies (prerequisites) and targets. The first match will stop the search for a given file, even if the same file exists in the directories that might still follow in the `VPATH` and are newer versions of the file than the one found first. This allows you to write `make` rules as if all files existed in the current directory.

With the example `VPATH=newsrc:src:obj`, the same rule (`file.o: file.c`) would be interpreted as if it was written like this:

```
obj/file.o: src/file.c
```

This example interpretation works only if `file.o` and `file.c` do not exist in the current directory, but are found in the `obj` and `src` directories, respectively, and `file.c` does not exist in the `newsrc` directory and `file.o` does not exist in the `newsrc` or `src` directories.

Use caution when writing `make` rules to allow `make` to look for dependencies and targets in alternate directories using the `VPATH` variable. Typically, there is nothing wrong with the following rule:

```
VPATH=dir
file.o: file.c
      $(CC) $(CFLAGS) -c file.c
```

when the file `file.c` is found in the current directory. The target `file.o` will be created in the current directory.

However, if `file.c` is found in the `dir` directory instead of the current directory, this example would be asking `make` to execute the following rule:

```
file.o: dir/file.c
      $(CC) $(CFLAGS) -c dir/file.c
```

Although make can adapt the dependency line (file.o: file.c) to the VPATH requirements, it cannot edit the command line. The command lines have to execute as written.

The make internal variables ($*, $@, $<, $?, $%) allow you to fully use the VPATH macro. These variables are correctly prefixed with the directory path coming from the VPATH search (if applicable). The following rule will execute correctly in all cases:

```
VPATH=dir
file.o: file.c
      $(CC) $(CFLAGS) -c -o $@ $<
```

For example, if both file.o and file.c are not found in the current directory, but exist in the dir directory, the internal variables will be set to the following, and the command line will be executed correctly:

```
$@ = dir/file.o
$< = dir/file.c
```

To take full advantage of the VPATH macro, many implicit make rules have been augmented by adding the output file option to the compilation line, instead of using the compiler default output file. The new method is compatible with older code. The following characteristics are maintained for implicit rules:

- After execution of the make command, only the input (prerequisites) and the target files remain.
- Should execution of an implicit rule require generation of temporary files, such as when executing the SCCS command, lex, yacc, and so on, the temporary files are always created in the current (.) directory and removed at the end of the action. Thus, write permission for the user to the . directory is required for successful execution of make in these cases.

To examine one possible use of the VPATH macro, assume that all project sources are stored in the src directory and all object files (relocatables, binaries, and libraries) are stored in the obj directory. Three object directories have been created: ./obj_debug, ./obj_normal, and ./obj_optim. These directories contain binaries for the debug version, the standard version, and the final, highly optimized version, respectively.

You could set three aliases (the following example is in the ksh shell), setting yourself CFLAGS as appropriate:

```
alias maked='make CFLAGS=... VPATH=src:obj_debug'
alias maken='make CFLAGS=... VPATH=src:obj_normal'
alias makeo='make CFLAGS=... VPATH=src:obj_optim'
```

You would then place the makefile in the . directory and, by executing the appropriate alias, generate the version corresponding to the immediate needs. maked would generate a version suitable for use with the debugger and would place the binaries in the corresponding directory. Likewise, maken would generate the normal version in its own directory. All this would be done from the same sources without any change of the makefile or the current directory.

Alternately, you could copy several sources out of the src directory, modify them, and execute the same aliases. According to the VPATH search rules, modified versions in the . directory will be located before the versions in the src directory and will be used to generate the appropriate binary versions. When the modified sources become stable, they can be moved back to the src directory, where they are the new base for the development process. In this manner it is easy to store the same sources for different binaries of the same program for different purposes.

This example succeeds because make will place the target file on the same place where it found it using the VPATH search, or in the . directory if it didn't find it anywhere. Consequently, the first execution of make in the program will cause all targets (usually binaries) to be put into the . directory because they do not yet exist anywhere. The targets must be moved into the appropriate obj_... directory by hand. The process described above is consistent from then on.

The same is true if, during the program development process, you add several new source files. The corresponding binaries must be moved to the appropriate obj_... directory by hand after the first binaries were generated into the . directory.

The VPATH macro does not currently work with POSIX rules.

Multiprocessing

Compilations may be multiprocessed when the environment variable NPROC is set to a value greater than 1. NPROC determines the maximum number of processes to be run in the background. You can set NPROC to any value, but extremely large values can overload the system. If you set NPROC in the makefile, the value you specify will take precedence over other NPROC settings unless the -e option is used. Set NPROC=1 in a makefile to inhibit parallel processing.

To force synchronization, include a slash (/) in a prerequisite list. All targets preceding a slash in a prerequisite list are completed before anything following the slash begins. The make command line is not multiprocessed.

If a slash (/) is used to force multiprocessing and the root file system has a date newer than that of the target, the target is executed.

NOTES

File names with the characters =, :, and @ do not work.

Commands executed directly by the shell, notably cd(1), are ineffectual across newline characters in make.

You cannot build lib(file.o) from file.o.

EXIT STATUS

When the -q option is specified, the make utility exits with one of the following values:

0 Successful completion.

1 The target was not up-to-date.

>1 An error occurred.

When the -q option is not specified, the make utility exits with one of the following values:

0 Successful completion.

>0 An error occurred.

FILES

/etc/MAKEFILE	File containing site default rules, read in before any user makefile
makefile	First file searched
Makefile	Second file searched
s.makefile	Third file searched
s.Makefile	Fourth file searched

SEE ALSO

ar(1), get(1), cd(1), lex(1), sh(1), yacc(1)

cc(1) in the *Cray Standard C Reference Manual*, Cray Research publication SR–2074

printf(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

Managing Projects with make, Talbott, Steve and Oram, Andrew, O'Reilly & Associates, Inc., 1991.

NAME

makekey – Generates encryption key

SYNOPSIS

/usr/lib/makekey

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The makekey utility improves the usefulness of encryption schemes that depend 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.

The first 8 input bytes (the *input key*) can be arbitrary ASCII characters. The last 2 characters (the *salt*) are best chosen from the set of digits, ., /, uppercase letters, and lowercase letters. The salt characters are repeated as the first 2 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 4096 cryptographic machines, which are all based on the National Bureau of Standards DES algorithm, but broken in 4096 various ways. Using the *input key* as key, a constant string is fed into the machine and recirculated several times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

The makekey utility is intended for programs that perform encryption (such as ed(1), vi(1), and crypt(1)). Usually, its input and output are pipes.

NOTES

Inclusion of the Data Encryption Standard (DES) encryption code requires a special license for sites outside the United States and Canada. If these encryption functions are unavailable on your system, check with your system administrator or site analyst.

SEE ALSO

crypt(1), ed(1), vi(1)