LynxOS Reference Manual

LynxOS Version 2.5

DOC-0170

Utility Programs (P–Z)

Printed in the United States of America.

Contents

Utility Programs (P-Z)

Utility Programs (P-Z)

NAME

`passwd` – change user password

SYNOPSIS

passwd [*user* **]**

DESCRIPTION

`passwd` replaces the current password recorded in the `/etc/passwd` file for the current user or for the user named *user*. It will prompt the user once for his current password, and then twice for a new one to guard against typographic errors. New passwords must be a minimum of 4 characters long if many different characters are used, 6 if the user decides to use a dull, monocase password, however if a user persists a passwd may be less then 4 characters but security will be compromised.

Passwords are kept in the publicly readable `/etc/passwd` file in an encrypted form.

FILES

`/etc/passwd`	User names and passwords
`/etc/ptemp`	Temporary copy of `/etc/passwd`

SEE ALSO

Utility Programs - `login`

Library Functions - `crypt(), getpass()`

NAME

`paste` – merge lines of one or more text files

SYNOPSIS

paste *file1 file2 ...*

paste -d *list file1 file2 ...*

paste -s [-d *list***]** *file1 file2 ...*

DESCRIPTION

In the first two forms, `paste` concatenates corresponding lines of the given input files *file1, file2*, etc. It treats each file as a column in a table and pastes them together horizontally (parallel merging). In the last form, `paste` treats each file independently, concatenating subsequent lines from each input file (serial merging). In all cases, lines are glued together with the tab character, or with characters from an optionally specified *list*. A file name of "**.**" refers to standard input. Output is sent to the standard output, so it can be piped or used as a filter.

OPTIONS

-d *list* Without this option, the new-line characters of each but the last file (or last line in case of the **-s** option) are replaced by tab characters. This option allows using the characters in list as replacement characters instead of tabs. The list is used circularly, i.e., when it is exhausted, it is reused. In parallel merging (i.e., no **-s**) option), the lines from the last file are always terminated with a new- line character, not from *list*. The list may contain the following special escape sequences: `\n` (new- line), `\t` (tab), `\` (backslash), and `\0` (null). Quoting may be necessary if characters have special meaning to the shell.

-s Merge subsequent lines rather than one from each input file: All lines of *file1* are concatenated together, followed by a new-line; then all lines of *file2* are concatenated together, followed by a new-line, etc. Use tab for concatenation, unless a *list* is specified with the **-d** option. Regardless of the *list*, the last character of each file is forced to be a new-line.

SEE ALSO

Utility Programs - `cut, grep`

NAME

`patch` – apply a diff file to an original

SYNOPSIS

patch [*options***] [***origfile* **[***patchfile***]] [+ [***options***] [***origfile***]]...**

but usually just

patch <patchfile

DESCRIPTION

`patch` will take a patch file containing any of the four forms of difference listing produced by the program and apply those differences to an original file, producing a patched version. By default, the patched version is put in place of the original, with the original file backed up to the same name with the extension `.orig` (~ on systems that do not support long file names), or as specified by the **-b** (**--suffix**), **-B** (**--prefix**), or **-V** (**--version-control**) options. The extension used for making backup files may also be specified in the `SIMPLE_BACKUP_SUFFIX` environment variable, which is overridden by the above options.

If the backup file already exists, `patch` creates a new backup file name by changing the first lowercase letter in the last component of the file's name into uppercase. If there are no more lowercase letters in the name, it removes the first character from the name. It repeats this process until it comes up with a backup file that does not already exist.

You may also specify where you want the output to go with a **-o** (**--output**) option; if that file already exists, it is backed up first.

If *patchfile* is omitted, or is a hyphen, the patch will be read from standard input.

Upon startup, patch will attempt to determine the type of the diff listing, unless over-ruled by a **-c** (**--context**), **-e** (**--ed**), **-n** (**--normal**), or **-u** (**--unified**) option. Context diffs (old-style, new-style, and unified) and normal diffs are applied by the `patch` program itself, while `ed` diffs are simply fed to the `ed` editor via a pipe.

`patch` will try to skip any leading garbage, apply the diff, and then skip any trailing garbage. Thus you could feed an article or message containing a diff listing to `patch`, and it should work. If the entire diff is indented by a consistent amount, this will be taken into account.

With context diffs, and to a lesser extent with normal diffs, `patch` can detect when the line numbers mentioned in the patch are incorrect, and will attempt to find the correct place to apply each hunk of the patch. As a first guess, it takes the line number mentioned for the hunk, plus or minus any offset used in applying the previous hunk. If that is not the correct place, `patch` will scan both forwards and backwards for a set of lines matching the context given in the hunk. First `patch` looks for a place where all lines of the context match. If no such place is found, and it's a context diff, and the maximum fuzz factor is set to 1 or more, then another scan takes place ignoring the first and last line of context. If that fails, and the maximum fuzz factor is set to 2 or more, the first two and last two lines of context are ignored, and another scan is made. (The default maximum fuzz factor is 2.) If `patch` cannot find a place to install that hunk of the patch, it will put the hunk out to a reject file, which normally is the name of the output file plus `.rej` (`#` on systems that do not support long file names). (Note that the rejected hunk will come out in context diff form whether the input patch was a context diff or a normal diff. If the input was a normal diff, many of the contexts will simply be null.) The line numbers on the hunks in the reject file may be different than in the patch file: they reflect the approximate location patch thinks the failed hunks belong in the new file rather than the old one.

As each hunk is completed, you will be told whether the hunk succeeded or failed, and which line (in the new file) `patch` thought the hunk should go on. If this is different from the line number specified in the diff you will be told the offset. A single large offset MAY be an indication that a hunk was installed in the wrong place. You will also be told if a fuzz factor was used to make the match, in which case you should also be slightly suspicious.

If no original file is specified on the command line, `patch` will try to figure out from the leading garbage what the name of the file to edit is. In the header of a context diff, the file name is found from lines beginning with `***` or `---`, with the shortest name of an existing file winning. Only context diffs have lines like that, but if there is an `Index:` line in the leading garbage, `patch` will try to use the file name from that line. The context diff header takes precedence over an Index line. If no file name can be intuited from the leading garbage, you will be asked for the name of the file to patch.

If the original file cannot be found or is read-only, but a suitable SCCS or RCS file is handy, `patch` will attempt to get or check out the file.

Additionally, if the leading garbage contains a `Prereq:` line, `patch` will take the first word from the prerequisites line (normally a version number) and check the input file to see if that word can be found. If not, `patch` will ask for confirmation before proceeding.

The upshot of all this is that you should be able to say, while in a news interface, the following:

```
| patch -d /usr/src/local/blurfl
```

and patch a file in the `blurfl` directory directly from the article containing the patch.

If the patch file contains more than one patch, `patch` will try to apply each of them as if they came from separate patch files. This means, among other things, that it is assumed that the name of the file to patch must be determined for each diff listing, and that the garbage before each diff listing will be examined for interesting things such as file names and revision level, as mentioned previously. You can give options (and another original file name) for the second and subsequent patches by separating the corresponding argument lists by a *(L'+*(R'. (The argument list for a second or subsequent patch may not specify a new patch file, however.)

`patch` recognizes the following options:

-b *suff*, **--suffix=***suff*
: causes *suff* to be interpreted as the backup extension, to be used in place of `.orig` or `~`.

-B *pref*, **--prefix=***pref*
: causes *pref* to be interpreted as a prefix to the backup file name. If this argument is specified, any argument from **-b** will be ignored.

-c, --context
: forces `patch` to interpret the patch file as a context diff.

-d *dir*, **--directory=***dir*
: causes `patch` to interpret *dir* as a directory, and `cd` to it before doing anything else.

-D *sym*, **--ifdef=***sym*
: causes `patch` to use the "`#ifdef`...`#endif`" construct to mark changes. *sym* will be used as the differentiating symbol.

-e, --ed
: forces `patch` to interpret the patch file as an `ed` script.

-E, --remove-empty-files
: causes `patch` to remove output files that are empty after the patches have been applied.

-f, --force
: forces `patch` to assume that the user knows exactly what he or she is doing, and to not ask any questions. It assumes the following: skip patches for which a file to patch can't be found; patch files even though they have the wrong version for the "Prereq:" line in the patch; and assume that patches are not reversed even if they look like they are. This option does not suppress commentary; use **-s** for that.

-t, --batch
: similar to **-f**, in that it suppresses questions, but makes some different assumptions: skip patches for which a file to patch can't be found (the same as **-f**); skip patches for which the file has the wrong version for the "Prereq:" line in the patch; and assume that patches are reversed if they look like they are.

-F *number*, **--fuzz=***number*
: sets the maximum fuzz factor. This option only applies to context diffs, and causes `patch` to ignore up to that many lines in looking for places to install a hunk. Note that a larger fuzz factor increases the odds of a faulty patch. The default fuzz factor is 2, and it may not be set to more than the number of lines of context in the context diff, ordinarily 3.

-l, --ignore-whitespace
: causes the pattern matching to be done loosely, in case the tabs and spaces have been munged in your input file. Any sequence of whitespace in the pattern line will

match any sequence in the input file. Normal characters must still match exactly. Each line of the context must still match a line in the input file.

-n, --normal forces `patch` to interpret the patch file as a normal diff.

-N, --forward causes `patch` to ignore patches that it thinks are reversed or already applied. See also **-R** .

-o *file*, **--output=***file*
causes *file* to be interpreted as the output file name.

-p[*number***], --strip[=***number***]**
sets the pathname strip count, which controls how pathnames found in the patch file are treated, in case the you keep your files in a different directory than the person who sent out the patch. The strip count specifies how many slashes are to be stripped from the front of the pathname. (Any intervening directory names also go away.) For example, supposing the file name in the patch file was

```
/u/howard/src/blurfl/blurfl.c
```

setting **-p** or **-p0** gives the entire pathname unmodified, **-p1** gives

```
u/howard/src/blurfl/blurfl.c
```

without the leading slash, **-p4** gives

```
blurfl/blurfl.c
```

and not specifying **-p** at all just gives you "`blurfl.c`", unless all of the directories in the leading path (`u/howard/src/blurfl`) exist and that path is relative, in which case you get the entire pathname unmodified. Whatever you end up with is looked for either in the current directory, or the directory specified by the **-d** option.

-r *file*, **--reject-file=***file*
causes *file* to be interpreted as the reject file name.

-R, --reverse tells `patch` that this patch was created with the old and new files swapped. (Yes, I'm afraid that does happen occasionally, human nature being what it is.) `patch` will attempt to swap each hunk around before applying it. Rejects will come out in the swapped format. The **-R** option will not work with `ed` diff scripts because there is too little information to reconstruct the reverse operation.

If the first hunk of a patch fails, `patch` will reverse the hunk to see if it can be applied that way. If it can, you will be asked if you want to have the **-R** option set. If it can't, the patch will continue to be applied normally. (Note: this method cannot detect a reversed patch if it is a normal diff and if the first command is an append (i.e. it should have been a delete) since appends always succeed, due to the fact that a null context will match anywhere. Luckily, most patches add or change lines rather than delete them, so most reversed normal diffs will begin with a delete, which will fail, triggering the heuristic.)

-s, --silent, --quiet
makes `patch` do its work silently, unless an error occurs.

-S, --skip causes `patch` to ignore this patch from the patch file, but continue on looking for the next patch in the file. Thus

patch -S + -S + <patchfile

will ignore the first and second of three patches.

-u, --unified forces `patch` to interpret the patch file as a unified context diff (a unidiff).

-v, --version causes `patch` to print out its revision header and patch level.

-V *method*, **--version--control=***method*
causes `method` to be interpreted as a method for creating backup file names. The type of backups made can also be given in the `VERSION_CONTROL` environment variable, which is overridden by this option. The **-B** option overrides this option, causing the prefix to always be used for making backup file names. The value of the `VERSION_CONTROL` environment variable and the

argument to the **-V** option are like the GNU Emacs 'version-control' variable; they also recognize synonyms that are more descriptive. The valid values are (unique abbreviations are accepted):

`t`
`numbered` Always make numbered backups.

`nil`
`existing` Make numbered backups of files that already have them, simple backups of the others. This is the default.

`never`
`simple` Always make simple backups.

-x *number*, **--debug=***number*
sets internal debugging flags, and is of interest only to `patch` patchers.

AUTHOR

Larry Wall <lwall@netlabs.com> along with many other contributors.

ENVIRONMENT

`TMPDIR` Directory to put temporary files in; default is `/tmp`.

`SIMPLE_BACKUP_SUFFIX`
Extension to use for backup file names instead of `.orig` or `~`.

`VERSION_CONTROL`
Selects when numbered backup files are made.

FILES

`$TMPDIR/patch*`

SEE ALSO

Utility Programs - `diff`

NOTES FOR PATCH SENDERS

There are several things you should bear in mind if you are going to be sending out patches. First, you can save people a lot of grief by keeping a patchlevel.h file which is patched to increment the patch level as the first diff in the patch file you send out. If you put a Prereq: line in with the patch, it won't let them apply patches out of order without some warning. Second, make sure you've specified the file names right, either in a context diff header, or with an Index: line. If you are patching something in a subdirectory, be sure to tell the patch user to specify a **-p** option as needed. Third, you can create a file by sending out a diff that compares a null file to the file you want to create. This will only work if the file you want to create doesn't exist already in the target directory. Fourth, take care not to send out reversed patches, since it makes people wonder whether they already applied the patch. Fifth, while you may be able to get away with putting 582 diff listings into one file, it is probably wiser to group related patches into separate files in case something goes haywire.

DIAGNOSTICS

Too many to list here, but generally indicative that `patch` couldn't parse your patch file.

The message `Hmm...` indicates that there is unprocessed text in the patch file and that `patch` is attempting to intuit whether there is a patch in that text and, if so, what kind of patch it is.

`patch` will exit with a non-zero status if any reject files were created. When applying a set of patches in a loop it behooves you to check this exit status so you don't apply a later patch to a partially patched file.

CAVEATS

`patch` cannot tell if the line numbers are off in an `ed` script, and can only detect bad line numbers in a normal diff when it finds a `change` or a `delete` command. A context diff using fuzz factor 3 may have the same problem. Until a suitable interactive interface is added, you should probably do a context diff in these cases to see if the changes made sense. Of course, compiling without errors is a pretty good indication that the patch worked, but not always.

`patch` usually produces the correct results, even when it has to do a lot of guessing. However, the results are guaranteed to be correct only when the patch is applied to exactly the same version of the file that the patch was generated from.

BUGS

Could be smarter about partial matches, excessively deviant offsets and swapped code, but that would take an extra pass.

If code has been duplicated (for instance with `#ifdef` *OLDCODE* ... `#else` ... `#endif`), `patch` is incapable of patching both versions, and, if it works at all, will likely patch the wrong one, and tell you that it succeeded to boot.

If you apply a patch you've already applied, `patch` will think it is a reversed patch, and offer to un-apply the patch. This could be construed as a feature.

NAME

`path` – find path to a file

SYNOPSIS

path [–P *path* **] [–** *mode* **]** *name* **...**

DESCRIPTION

`path` searches through a list of directories to find entries matching the given *name*s which are accessible in the given *mode*. The list of directories is taken from the environment variable `PATH` unless another path is given. If no mode is given, files are checked for existence only. The completed pathname for each *name* is printed as soon as an occurrence is found in a directory, and no more directories are searched for that *name*.

OPTIONS

–P *path* — The list of directories is taken from *path* instead of the `PATH` environment variable. The list should be in the same format as `PATH`; that is, separated by colons, no intervening spaces, with empty fields representing the current directory.

– *mode* — The *mode* given is used to check for files. *mode* can be any combination of the characters `r`, `w`, and `x`, meaning read, write, and execute. Note that directory files may appear to be executable files.

SEE ALSO

Utility Programs - `find, ls`

System Calls - `access()`

Library Functions – `findpath()`

NAME

`pfa_dsm` – disassemble a pfa dumped text

SYNOPSIS

pfa_dsm *kernel dumpedtext* **[-o** *modulename* **[** *modulename ...* **]]**
[-d *routine* **[** *routine ...* **]]**

DESCRIPTION

`pfa` collects the information about what portion of the operating system code has been executed. It works by booting a specific kernel that loads the target kernel in high memory and overwriting the kernel text segment with illegal instructions. When a fault comes, the code is copied from the high memory. Thus, the amount of operating system codes executed can be measured by analyzing the text segment. *dumpedtext* is a dumped copy from `pfa_dump`.

kernel is the target kernel corresponding to *dumpedtext*. `pfa_dsm` compares *dumpedtext* with kernel and disassemble the kernel text.

If the code has been executed, the output line will begin with "1", otherwise, it will begin with **0**. You must specify **-o** or **-d**.

If **.o** module names are specified, the information printed will be based on each *routine*, internal or external, of the module. If routine names are specified, the information about the routines will be printed.

SEE ALSO

Utility Programs - `pfa_dump, pfa_list`

File Formats - `a.out`

NAME

pfa_reset – reset kernel text to breakpoint instruction

SYNOPSIS

pfa_reset *kernel* **[-o** *modulename* **[** *modulename ...* **]] [-d** *routine* **[** *routine ...* **]]**

DESCRIPTION

pfa collects the information about what portion of the operating system code has been executed. It works by booting a specific *kernel* that loads the target kernel in high memory and overwriting the kernel text segment with illegal instructions. When a fault comes, the code is copied from the high memory. Thus, the amount of operating system codes executed can be measured by analyzing the text segment.

kernel is the kernel file corresponding to the current running kernel. pfa_reset resets the running kernel text to breakpoint instructions based on the specified modules or routines. If **.o** module names are specified, the kernel code corresponding to the module will be reset to breakpoint instructions.

If routine names are specified, the kernel code corresponding to the *routine* will be reset to breakpoint instructions.

SEE ALSO

Utility Programs - pfa_dump, pfa_list

File Formats - a.out

NAME

`ping` – send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

ping [-dfnqrvR] [-c *count***] [-i** *wait***] [-l** *preload***] [-p** *pattern***] [-s** *packetsize***]**

DESCRIPTION

`ping` uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams (*pings*) have an IP and ICMP header, followed by a "`struct timeval`" and then an arbitrary number of pad bytes used to fill out the packet.

OPTIONS

-c *count* — Stop after sending (and receiving) *count* ECHO_RESPONSE packets.

-d — Set the SO_DEBUG option on the socket being used.

-f — Flood `ping`. Outputs packets as fast as they come back or one hundred times per second, whichever is more. For every ECHO_REQUEST sent a period "." is printed, while for every ECHO_REPLY received a backspace is printed. This provides a rapid display of how many packets are being dropped. Only the super-user may use this option. *This can be very hard on a network and should be used with caution.*

-i *wait* — Wait *wait* seconds between sending each packet. The default is to wait for one second between each packet. This option is incompatible with the **-f** option.

-l *preload* — If *preload* is specified, `ping` sends that many packets as fast as possible before falling into its normal mode of behavior.

-n — Numeric output only. No attempt will be made to lookup symbolic names for host addresses.

-p *pattern*	You may specify up to 16 "pad" bytes to fill out the packet you send. This is useful for diagnosing data-dependent problems in a network. For example, "`-p ff`" will cause the sent packet to be filled with all ones.
-q	Quiet output. Nothing is displayed except the summary lines at startup time and when finished.
-R	Record route. Includes the RECORD_ROUTE option in the ECHO_REQUEST packet and displays the route buffer on returned packets. Note that the IP header is only large enough for nine such routes. Many hosts ignore or discard this option.
-r	Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to `ping` a local host through an interface that has no route through it (e.g., after the interface was dropped by `routed`(1)).
-s *packetsize*	Specifies the number of data bytes to be sent. The default is 56, which translates into 64 ICMP data bytes when combined with the 8 bytes of ICMP header data.
-v	Verbose output. ICMP packets other than ECHO_RESPONSE that are received are listed.

USAGE

When using `ping` for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be "pinged". Round-trip times and packet loss statistics are computed. If duplicate packets are received, they are not included in the packet loss calculation, although the round trip time of these packets is used in calculating the minimum/average/maximum round-trip time numbers. When the specified number of packets have been sent (and received) or if the program is terminated with a `SIGINT`, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. Because of the load it can impose on the network, it is unwise to use `ping` during normal operations or from automated scripts.

ICMP PACKET DETAILS

An IP header without options is 20 bytes. An ICMP ECHO_REQUEST packet contains an additional 8 bytes worth of ICMP header followed by an arbitrary amount of data. When a *packetsize* is given, this indicated the size of this extra piece of data (the default is 56). Thus the amount of data received inside of an IP packet of type ICMP ECHO_REPLY will always be 8 bytes more than the requested data space (the ICMP header).

If the data space is at least eight bytes large, `ping` uses the first eight bytes of this space to include a timestamp which it uses in the computation of round trip times. If less than eight bytes of pad are specified, no round trip times are given.

DUPLICATE AND DAMAGED PACKETS

Ping will report duplicate and damaged packets. Duplicate packets should never occur, and seem to be caused by inappropriate link-level retransmissions. Duplicates may occur in many situations and are rarely (if ever) a good sign, although the presence of low levels of duplicates may not always be cause for alarm.

Damaged packets are obviously serious cause for alarm and often indicate broken hardware somewhere in the `ping` packet's path (in the network or in the hosts).

TRYING DIFFERENT DATA PATTERNS

The (inter)network layer should never treat packets differently depending on the data contained in the data portion. Unfortunately, data-dependent problems have been known to sneak into networks and remain undetected for long periods of time. In many cases the particular pattern that will have problems is something that doesn't have sufficient "transitions", such as all ones or all zeros, or a pattern right at the edge, such as almost all zeros. It isn't necessarily enough to specify a data pattern of all zeros (for example) on the command line because the pattern that is of interest is at the data link level, and the relationship between what you type and what the controllers transmit can be complicated.

This means that if you have a data-dependent problem you will probably have to do a lot of testing to find it. If you are lucky, you may manage to find a file that either can't be sent across your network or that takes much longer to transfer than other similar length files. You can then examine this file for repeated patterns that you can test using the **-p** option of `ping`.

TTL DETAILS

The TTL value of an IP packet represents the maximum number of IP routers that the packet can go through before being thrown away. In current practice you can expect each router in the Internet to decrement the TTL field by exactly one.

The TCP/IP specification states that the TTL field for TCP packets should be set to 60, but many systems use smaller values (4.3 BSD uses 30, 4.2 used 15).

The maximum possible value of this field is 255, and most Unix systems set the TTL field of ICMP ECHO_REQUEST packets to 255. This is why you will find you can "`ping`" some hosts, but not reach them with `telnet`(1) or `ftp`(1).

In normal operation `ping` prints the ttl value from the packet it receives. When a remote system receives a `ping` packet, it can do one of three things with the TTL field in its response:

- Not change it; this is what Berkeley Unix systems did before the 4.3BSD-Tahoe release. In this case the TTL value in the received packet will be 255 minus the number of routers in the round-trip path.
- Set it to 255; this is what current Berkeley Unix systems do. In this case the TTL value in the received packet will be 255 minus the number of routers in the path *from* the remote system *to* the pinging host.
- Set it to some other value. Some machines use the same value for ICMP packets that they use for TCP packets, for example either 30 or 60. Others may use completely wild values.

BUGS

Many Hosts and Gateways ignore the RECORD_ROUTE option.

The maximum IP header length is too small for options like RECORD_ROUTE to be completely useful. There's not much that can be done about this, however.

Flood pinging is not recommended in general, and flood pinging the broadcast address should only be done under very controlled conditions.

SEE ALSO

Utility Programs - `netstat`, `ifconfig`, `routed`

HISTORY

The `ping` command appeared in 4.3BSD.

NAME

`portmap` – DARPA port to RPC program number mapper

SYNOPSIS

`portmap`

DESCRIPTION

`portmap` is a server that converts RPC program numbers into DARPA protocol port numbers. It must be running in order to make RPC calls.

When an `RPC` server is started, it will tell `portmap` what port number it is listening to, and what RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it will first contact portmap on the server machine to determine the port number where RPC packets should be sent.

NOTE

If `portmap` crashes, all servers must be restarted.

SEE ALSO

Utility Programs - `rpcinfo`

NAME

`pr –` produce a formatted listing of a file

SYNOPSIS

pr [*options***]** *file ...*

DESCRIPTION

`pr` produces a paginated listing of one or more files. Each page begins with a heading, which by default contains the current date, the file name, and the page number. If no file arguments are given, standard input is used.

In addition to paginating files, `pr` will also translate tabs into strings of spaces and translate strings of the form “underscore-backspace-character” into standout mode escape sequences for the output terminal.

OPTIONS

– *n* Produce output consisting of *n* columns (default is 1 column).

–b *heading* Defines the format of the *heading*. The string *heading* is of the form

left'center'right

The *left* part is adjusted to the left margin, *center* is centered, and *right* is adjusted to the right margin. The strings may contain arbitrary text, along with the following special elements:

```
%f  Inserts current file name.
%p  Inserts current page number.
%d  Inserts current date.
%t  Inserts current time.
%'  Inserts the quote character.
%%  Inserts a percent sign.
```

The default title string is:

```
Date: %d Time:%t'%f'Page %p
```

–f *n* Set the page footer length to *n*.

-h *n*	Set the page header length to *n*. If possible, the heading is printed on the second line of the page. If the header size is zero, no heading is printed.
-m	Output all *files* at the same time, one file per column.
-l	Use formfeeds instead of newlines to separate pages. (This option does not change the effective page length).
-p *n*	Set the length of a page to *n* lines (default is 66).
-s *c*	Separate the columns by the character *c* instead of by spaces.
-t *n*	Set the tab size to *n*. If *n* is zero, tabs will not be expanded.
-w *n*	Set the page width to *n* characters rather than the default 80. This option is useful when multi-column output is active.

SEE ALSO

Utility Programs - `cat, spool`

NAME

`preboot` – kernel loader program

DESCRIPTION

Please refer to the platform-specific manual page for `preboot`. For example, if you use LynxOS for x86, you can access the correct `preboot` manual page by entering:

`man preboot_x86`

SEE ALSO

Utility Programs - `preboot_x86`, `preboot_ppc`, `preboot_68k`, `preboot_sparc`

NAME

`preboot` – kernel loader program (68k)

SYNOPSIS

`[bR] ?`

`R` *major minor*

`B -[afmNr] [xy[z]] [/`*file*`]`

DESCRIPTION

`preboot` is a kernel loader that provides an interface to the user prior to kernel boot-up.

The `preboot` interface allows the user to change the default root file system and other boot defaults. If there is no keyboard input for 10 seconds after boot-up, `preboot` will automatically boot from the default root file system. The default root file system is set from the command `makeboot`; see the `makeboot(1)` manual page for more information.

Type a **`?`** at the preboot prompt for help. To print the default settings for the root or boot device, precede the **`?`** with the appropriate flag. For example, to print the defaults for the root file system, type:

`R ?`

OPTIONS

`?`	Help; if preceded by **`b`** or **`R`**, will print the default values for that command.
`R` *major minor*	Set root device to *major,minor*.
`B`	Set kernel and boot flags. See “Boot Flags” for more information.

Boot Flags

The syntax for setting the kernel and boot flags is:

B –[afmNr] [xy[z]] [/*file*]

The boot flag defaults are set from a previous reboot. Specifying these options from `preboot` will override the defaults.

–	single-user mode	
a	multi-user mode	
f	don't execute `fsck` for file system checks	
m	execute `fsck` to check the file system integrity	
N	make nodes	
r	mount root file system in read-only mode	
x	The device type is specified by one of the following values for *x*:	
	s	SCSI devices
y	The unit number is specified by one of the following values for *y*:	
	0–6	SCSI disk devices
z	The partition is specified by one of the following values for *z*:	
	a	Partition A
	b	Partition B
	c	Partition C
	d	Partition D
/*file*	The kernel file is specified by *file* as an absolute pathname. The default is `/lynx.os`.	

EXAMPLES

Setting the Root Device

The root device is the location of a LynxOS file system. To display the current default root device, type the following at the `preboot` prompt:

```
R ?
```

Note that on 68k systems, `preboot` displays the value of the root device hexadecimal notation.

To set the root device to something other than the default, use:

R *major minor*

The major and minor number of a device map to a definition of that device in the kernel. On LynxOS 68k systems, the major number, *major*, is 0 for SCSI devices. The minor number, *minor*, reflects the SCSI ID and partition. Default values for LynxOS 68k systems are shown in the table “Minor Number Values for LynxOS SCSI Devices (in decimal)”.

Table 1: Minor Number Values for LynxOS SCSI Devices (in decimal)

SCSI ID	Partition a	Partition b	Partition c	Partition d
0	16	32	48	64
1	17	33	49	65
2	18	34	50	66
3	19	35	51	67
4	20	36	52	68
5	21	37	53	69
6	22	38	54	70

For example, to set the root device for a disk with SCSI ID 2 and partition B, type the following at the `preboot` prompt:

```
R 0 34
```

Setting the boot device

The boot device is the location of the LynxOS kernel that will be booted. To set the boot device to partition B of a SCSI device at ID 2, type the following:

```
B s2b
```

See “Boot Flags” for options that can be used with the **B** command.

NOTE

`preboot` can only be used with the `makeboot` utility.

The major and minor numbers recognized by `preboot` correspond to those used by *file*. The kernel will crash/hang if an incorrect boot device is specified.

SEE ALSO

Utility Programs - `makeboot, reboot`

NAME

`preboot` – kernel loader program (PowerPC)

SYNOPSIS

[bR] ?

R *major minor*

C *logical_unit*

b -[afmNr] [xy[z]] [/*file***]**

q

DESCRIPTION

`preboot` is a kernel loader that provides an interface to the user prior to kernel boot-up.

The `preboot` interface allows the user to change the default root file system and other boot defaults. If there is no keyboard input for 10 seconds after boot-up, `preboot` will automatically boot from the default root file system. The default root file system is set from the command `makeboot`; see the `makeboot(1)` manual page for more information.

Type a **?** at the preboot prompt for help. To print the default settings for the root or boot device, precede the **?** with the appropriate flag. For example, to print the defaults for the root file system, type:

```
R ?
```

OPTIONS

?	Help; if preceded by **b** or **R**, will print the default values for that command.
R *major minor*	Set root device to *major,minor*. By default, this is the same as the boot device.
C *logical_unit*	Set controller logical unit number (CLUN) to *logical_unit* for the boot controller.

b	Set kernel and boot flags. See "Boot Flags" for more information.
q	Exit to PPC Bug monitor.

Boot Flags

The syntax for setting the kernel and boot flags is:

`b -[afmNr] [xy[z]] [/`*file*`]`

The boot flag defaults are set from a previous reboot. Specifying these options from `preboot` will override the defaults.

-	single-user mode
a	multi-user mode
f	don't execute `fsck` for file system checks
m	execute `fsck` to check the file system integrity
N	make nodes
r	mount root file system in read-only mode
x	The device type is specified by one of the following values for *x*:
	s all bootable devices
y	The unit number is specified by one of the following values for *y*:
	0–1 floppy disk devices
	0–4 IDE disk devices
	0–6 SCSI devices
z	The partition is specified by one of the following values for *z*:
	a Partition A
	b Partition B
	c Partition C

	d Partition D
/file	The kernel file is specified by *file* as an absolute pathname. The default is `/lynx.os`.

EXAMPLES

Setting the Root File System

The root device is the location of a LynxOS file system. To display the current default root device, type the following at the `preboot` prompt:

R ?

To set the root device to something other than the default, use:

R *major minor*

The major and minor number of a device map to a definition of that device in the kernel. On LynxOS PowerPC systems, the default major device number, *major*, values are shown in the table “Default Major Number Values for LynxOS 2.5 Devices”.

Table 1: Default Major Number Values for LynxOS 2.5 Devices

ID	Description
0	RAM disk
1	Floppy 0-3
2	SCSI disk (includes CD-ROM)
3	SCSI tape

NOTE: LynxOS treats CD-ROM media as read-only disks.

The minor number, *minor*, reflects the device ID and partition. Default values for LynxOS PowerPC systems are shown in the table “Minor Number Values for LynxOS 2.5 Devices”.

Table 2: Minor Number Values for LynxOS 2.5 Devices

Type		ID	Whole Disk	Partition a	Partition b	Partition c	Partition d
SCSI	IDE	**0**	0	16	32	48	64
		1	1	17	33	49	65
		2	2	18	34	50	66
		3	3	19	35	51	67
		4	4	20	36	52	68
		5	5	21	37	53	69
		6	6	22	38	54	70

To set the root device for partition B of a SCSI disk at ID 2, type the following at the `preboot` prompt:

```
R 2 34
```

Setting the Boot Device

The boot device is the location of the LynxOS kernel that will be booted. To set the boot device to partition B of a SCSI device at ID 2, type the following:

```
b s2b
```

See “Boot Flags” for options that can be used with the **b** command.

Setting the Controller Logical Unit Number

LynxOS `preboot` inherits the Controller Logical Unit Number (CLUN) that is set by the PowerPC firmware with the command `pboot`.

```
xxx-Bug> pboot CLUN xxx
```

This CLUN is different for SCSI, floppy, and IDE devices. For example, to boot from a floppy, use the command:

```
xxx-Bug> pboot 1 0
```

However, you may want to boot a floppy with LynxOS `preboot` then use `preboot` to load LynxOS from a SCSI disk in the system. In this case, you'll need to change the CLUN number from **1** (floppy) to **0** (SCSI device) with the **C** command in preboot.

You can determine the CLUN of all the devices on your system with the `ioi` command at the VMEBug prompt. For example:

```
xxx-Bug> ioi
I/O Inquiry Status:
CLUN DLUN  CNTRL-TYPE  DADDR DTYPE  RM  Inquiry-Data
0    50    NCR53C810   5     $05    Y   TEAC CD-ROM CD-56S 1.0B
0    60    NCR53C810   6     $00    N   IBM DSAS-3540 !W S47Q
1     0    PC8477      0     $00    Y   <None>
```

In this example, the controller logical unit number (CLUN) is 0 for SCSI devices 50 and 60. You can use the **C** command at the `preboot` prompt to change the CLUN to 0 if you want to boot from disk 60. For example:

```
C 0
```

COMPATIBILITY ISSUES

`preboot` was implemented on the PowerPC platform in LynxOS 2.5, introducing some some compatibility issues with previous releases of LynxOS. To boot a pre-2.5 kernel, you cannot use the **R** command to set the root device because pre-LynxOS 2.5 kernels cannot communicate with `preboot`. You must use the facilities of the PPC Bug to set the root device when using `preboot` to boot pre-LynxOS 2.5 kernels on the PowerPC.

For example, assume that a SCSI disk at ID 0 is configured as follows:

Partition B —LynxOS 2.4.0	Partition C—LynxOS 2.5.0 (with preboot)

In this case, `preboot` on partition C has been made bootable with the `makeboot` utility (see `makeboot(1)` for more information).

To boot the LynxOS 2.5 kernel `/lynx.os` on partition C:

1. Boot preboot from the PPC Bug.

   ```
   Bug> pboot 0 60
   ```

2. Press **Enter** at the `preboot` prompt.

To boot the LynxOS 2.4 kernel /lynx.os on partition B of a SCSI disk at ID 0:

1. From PPC Bug, set the major device number (in hexidecimal) of the root device.

   ```
   Bug> mm 80000074; B
   80000074 10 ? 15
   80000075 10 ? 00
   80000076 10 ?
   80000077 02 ? 0          major number in hex
   80000078 10 ? .
   ```

2. Set the minor device number (in hexidecimal) of the root device.

   ```
   Bug> mm 80000074; B
   80000074 10 ? 14
   80000075 10 ? 00
   80000076 10 ?
   80000077 20 ? 20         minor number in hex
   80000078 10 ? .
   ```

3. Boot `preboot`.

   ```
   Bug> pboot 0 32
   ```

4. At the `preboot` prompt, boot the LynxOS 2.4 kernel.

   ```
   b s0b /lynx.os
   ```

NOTE

IDE/EIDE controllers are not fully supported in this release.

`preboot` can only be used with the `makeboot` utility.

SEE ALSO

Utility Programs - `makeboot`, `reboot`

NAME

`preboot` – kernel loader program (SPARC)

SYNOPSIS

b **[***boot_device***]** **[***boot_file***]**

f *boot_device boot_file*

r *major minor*

DESCRIPTION

`preboot` is the secondary boot loader which is used to load LynxOS on a machine which has the Open Boot PROM. `preboot` is installed on a bootable floppy or hard disk using `makeboot`. The disk must be labeled as a SunOS-compatible disk using special arguments to `mkpart`. Disks without a SunOS-style disk label cannot be accessed by of the Open Boot PROM. For complete information on labeling the disk, see `mkpart`, SPARC version.

When a machine is booted using `preboot`, the user has many choices which can be entered at the console. If nothing is entered in ten seconds, `preboot` will attempt to boot the machine using one default boot device and file. These defaults are stored in the Open Boot PROM variables *boot_device* and *boot_file*, and can be altered by the user. For more information, see the Open Boot PROM Users' Guide.

OPTIONS

b [*boot_device*] [*boot_file*]
: This is used to boot the given file from the given device (which must be a SCSI disk). Device names are given as Open Boot PROM names; e.g. `disk3` or `/sbus/esp@0,800000/sd@0,0`. If either or both parameters are omitted, preboot will substitute the appropriate default. The **b** command only works on SCSI disks, not on floppy disks.

f *boot_device boot_file*
: This is used to boot from a floppy disk. The default floppy device is called `/fd` or `floppy`. The **f** command only works on floppies.

r *major minor*
: This is used to set the major/minor numbers of the root device which LynxOS will mount when it comes up. By default, the root device is the device on which `preboot` is installed. Typing **r ?** will cause preboot to print the current root device numbers.

The `preboot` program does not recognize the names from LynxOS x86 or 68k systems. SPARC systems use the naming scheme provided by, and required by, the Open Boot PROM.

SEE ALSO

Utility Program - `mkpart, makeboot, sparcboot`

Documents - *Open Boot PROM Users' Guide*

NAME

`preboot` – kernel loader program (x86)

SYNOPSIS

`[bR] ?`

`R` *major minor*

`b -[afmNr] [xy[z]] [/`*file***`]`**

DESCRIPTION

`preboot` is a kernel loader that provides an interface to the user prior to kernel boot-up. Without `preboot`, a kernel of size greater than 512 KB will not be able to boot. `preboot` can be used to boot a kernel of any size.

The `preboot` interface allows the user to change the default root file system and other boot defaults. If there is no keyboard input for 10 seconds after boot-up, `preboot` will automatically boot from the default root file system. The default root file system is set from the command `makeboot`; see the `makeboot(1)` manual page for more information.

Type a **`?`** at the preboot prompt for help. To print the default settings for the root or boot device, precede the **`?`** with the appropriate flag. For example, to print the defaults for the root file system, type:

```
R ?
```

OPTIONS

`?`	Help; if preceded by **`b`** or **`R`**, will print the default values for that command.
`R` *major minor*	Set root device to *major,minor*. By default, this is the same as the boot device.
`b`	Set the boot device and/or boot flags. See “Boot Flags” for more information.

Boot Flags

The syntax for setting the kernel and boot flags is:

`b -[afmNr] [xy[z]] [/`*file*`]`

The boot flag defaults are set from a previous reboot. Specifying these options from `preboot` will override the defaults.

-	single-user mode
a	multi-user mode
f	don't execute `fsck` for file system checks
m	execute `fsck` to check the file system integrity
N	make nodes
r	mount root file system in read-only mode
x	The device type is specified by one of the following values for *x*:

f	floppy disk devices
h	IDE devices
s	SCSI devices

y	The unit number is specified by one of the following values for *y*:

0–1	floppy disk devices
0–3	IDE disk devices where:

0	primary master
1	primary slave
2	secondary master
3	secondary slave

0–6	SCSI disk devices

z The partition is specified by one of the following values for *z*:

a	Partition A
b	Partition B
c	Partition C
d	Partition D

/file The kernel file is specified by *file* as an absolute pathname. The default is `/lynx.os`.

EXAMPLES

Setting the Root Device

The root device is the location of a LynxOS file system. To display the current default root device, type the following at the `preboot` prompt:

R ?

To set the root device to something other than the default, use:

R *major minor*

The major and minor number of a device map to a definition of that device in the kernel. On LynxOS x86 systems, the default major device number, *major*, values are shown in the table "Default Major Number Values for LynxOS 2.5 Devices".

Table 1: Default Major Number Values for LynxOS 2.5 Devices

ID	Description
0	IDE/EIDE
1	Floppy 0-3
2	Adaptec 1522 SCSI
3	Adaptec 1542 SCSI
4	Adaptec 1742 SCSI

Table 1: Default Major Number Values for LynxOS 2.5 Devices

ID	Description
5	Adaptec 2742 SCSI
6	Adaptec 2940 SCSI

The minor number, *minor*, reflects the device ID and partition. Default values for LynxOS x86 systems are shown in the table “Minor Number Values for LynxOS 2.5 Devices”.

Table 2: Minor Number Values for LynxOS 2.5 Devices

Type		ID	Whole Disk	Partition a	Partition b	Partition c	Partition d
SCSI	IDE	**0**	0	16	32	48	64
SCSI	IDE	**1**	1	17	33	49	65
SCSI	IDE	**2**	2	18	34	50	66
SCSI	IDE	**3**	3	19	35	51	67
SCSI		**4**	4	20	36	52	68
SCSI		**5**	5	21	37	53	69
SCSI		**6**	6	22	38	54	70

To set the root device for partition B of an Adaptec 1522 SCSI disk at ID 2, type the following at the `preboot` prompt:

```
R 2 34
```

Setting the Boot Device

The boot device is the location of the LynxOS kernel that will be booted. To set the boot device to partition B of a SCSI device at ID 2, type the following:

```
b s2b
```

See “Boot Flags” for options that can be used with the **b** command.

NOTE

`preboot` can only be used with the `makeboot` utility.

The specified *file* name must start with **/** (i.e., `/lynx.os`).

The major and minor numbers recognized by `preboot` correspond to those used by *file*. The kernel will crash/hang if an incorrect boot device is specified.

SEE ALSO

Utility Programs - `makeboot, reboot`

NAME

`printenv` – print environment

SYNOPSIS

`printenv [-r] [` *var ...* **`]`**

DESCRIPTION

`printenv` prints the current values variables in the current environment list. If no *var* names are given, the entire list is printed. Non-printable characters in the environment are translated to printable sequences.

Each element of the environment list is a string of the form:

name=*value*

The environment of a process is inherited by each of its child processes. The shell allows its environment to be manipulated as shell variables.

OPTIONS

`-r`	Raw output mode. Translation of non-printable characters is suppressed.

SEE ALSO

Utility Programs - `dlsh`

System Calls - `execve()`

Library Functions - `getenv()`

NAME

`prio` – set real-time priority

SYNOPSIS

prio [+*incr*|-*incr*|*prio*] [*command*]

DESCRIPTION

`prio` allows the calling process to execute commands at particular priorities, or, if no command is given, to set the calling process' own priority to the indicated priority. The priority is determined in the following manner.

If the first argument to `prio` is a non-negative integer, that number is taken as the desired priority for the command or the caller.

If the first argument is of the form +*number*, that number is taken and added to the priority of the calling process to arrive at the new priority of either the command or the calling process. Likewise, if the first argument is -*number*, then the number is subtracted from the calling process' priority to arrive at the new priority.

Any *command* given as an additional argument to `prio` will be executed at the indicated priority. If no *command* is given, the priority of the calling process is set to the indicated priority.

If `prio` is called with a *command*, but with no priority argument, then the *command* is executed at a priority one less than the priority of the calling process.

If `prio` is called with no arguments at all, then the effect is as if one had called `prio -1`: the priority of the calling process is decremented.

EXAMPLES

Increase the priority of the calling process by `1`:

```
prio +1
```

Set the priority of the calling process to 255:

```
prio 25
```

Run `ls -lR` at priority 16:

```
prio 16 ls -lR
```

Run kermit at a priority 10 greater than the calling process:

```
prio +10 kermit
```

The following command is probably a mistake:

```
prio 200 data_aq < /dev/device | tee /tmp/contig_file | pid_ctl
> /dev/other_device
```

The intention is probably to run all of data `aq`, `tee`, and `pid_ctl` at priority 200. This is not what will happen. Because of the way processes are created by most shells, only `data_aq` will run at priority 200.

A more appropriate example would be:

```
prio 200 sh -c 'data_aq < /dev/device | tee /tmp/contig_file |
pid_ctl > /dev/other_device'
```

This command runs a shell at priority 200, which, in setting up the three subsidiary processes, will run them all at priority 200. Note that this example uses a quoting mechanism appropriate to `sh` or `csh`.

NAME

ps – display status of current processes

SYNOPSIS

ps [afmnostxFT] [r [*delay* **]] [-p** *pid* **] [-u** *uid* **] [-g** *pgrp* **]**

DESCRIPTION

ps displays the current status of processes in the system and shows at the bottom of its display total memory used by the processes and memory available (In the case of virtual memory it will show virtual memory available). The statistics available for each process are as follows:

pid	Process ID number.
tid	Thread ID number. If this thread is not the initial thread of this process, the thread ID number is followed by a plus sign (+).
ppid	Process ID of the parent process.
pgrp	Process group number.
pri	System priority.
text	Size of text (executable code) segment, in terms of 1KB units. If the text segment is being shared with another process displayed by ps, the text size is followed by an asterisk.
stack	Size of the process stack segment.
data	Size of the process data segment.
sem	Semaphore ID number on which process is waiting, if any.
time	User and system time used so far by the thread if option **t** (includes thread IDs) is present. Otherwise, user and system time used so far by the process, given as follows:

ss.ss Seconds (less than a minute)

	`mm:ss`	Minutes and seconds (less than an hour).
	`hh:mm:ss`	Hours, minutes and seconds (< 100 hours).
	`dd+hh:mm`	Days, hours, and minutes (< 100 days).
	`ddddd+hh`	Days and hours (at least 100 days).
`dev`	Control device of the process, as recorded in `/dev`.	
`flags`	Change value of process' (or thread's) flags.	
`user`	User name (based on effective user ID) of the process owner.	
`S`	Current process state, one of	
	`C`	Currently executing process (will usually be `ps` itself).
	`R`	Ready to run, but not running.
	`S`	Suspended by a signal (`SIGSTOP`, `SIGTSTP`, `SIGTTIN`, or `SIGTTOU`).
	`W`	Waiting on a semaphore for some resource.
	`B`	New process awaiting completion of its `fork` operation.
	`Z`	Zombie waiting to be cleaned up by the `init` process.
`name`	Pathname of the executing process or thread. Currently, the operating system only maintains the first 32 bytes of the pathname.	
`dlim`	Current data size resource limit, in terms of 1KB units.	
`d%`	Percentage of data size limit currently used.	
`slim`	Current stack size resource limit.	
`s%`	Percentage of stack size limit currently used.	
`smem`	Shared memory pages in use, and number of shared memory regions.	
`files`	Number of files currently open for this process or thread.	

By default, `ps` displays all processes with the same effective user ID as the current real user ID.

OPTIONS

a	Includes processes owned by others.
f	Substitute the `flags` field for the `dev` field
m	Show memory usage display instead of normal system use display. This option provides the `dlim`, `d%`, `slim`, `s%`, and `smem` fields but leaves out `sem`, `time`, `dev`, `user`, and `S`.
n	Includes the operating system "null" pseudo-process in the output. The accumulated time for the null process is an indication of system idle time.
o	Includes the supervisor stack size and process state segment size in the stack and data size values.
-p *pid*	Includes information for process *pid*.
-g *pgrp*	Includes information for all processes with process group number *pgrp*.
r *delay*	Tells `ps` to indefinitely repeat. The display (selected by the other options) is drawn at the top of the CRT. By default, `ps` waits 10 seconds between updates; *delay* may be given to increase or decrease the update frequency.
s	Substitute the `sem` field for the `time` field.
t	Includes thread IDs.
-u *uid*	Includes information for all processes owned by user *uid*.
x	Includes processes, owned by you, with no control device.
F	Show the number of files currently open for each process or thread.
T	Includes stream tasks.

FILES

`/dev/mem`	memory device, used to get information about processes.

SEE ALSO

Utility Programs - `kill, who`

System Calls - `info()`

DIAGNOSTICS

Since `ps` cannot sample and display the status off all processes at the speed at which the processes run, its accuracy is limited.

NAME

`pwd` – working directory name

SYNOPSIS

`pwd`

DESCRIPTION

`pwd` prints the path name of the working (current) directory.

SEE ALSO

Utility Programs - `cd`

Library Functions - `getwd()`

NAME

`query` – Query the `routed` daemon to get route information

SYNOPSIS

`query [-n]` *host...*

DESCRIPTION

`query` displays the current route information for the specfied hosts by contacting their `routed` daemons.

OPTIONS

`-n` Show network addresses as numbers (normally `query` interprets addresses and attempts to display them symbolically).

SEE ALSO

Utility Programs - `routed`

NAME

`ranlib` – generate index to archive.

SYNOPSIS

`ranlib [-v|-V]` ***`archive`***

DESCRIPTION

`ranlib` generates an index to the contents of an archive, and stores it in the archive. The index lists each symbol defined by a member of an archive that is a relocatable object file.

You may use

```
nm -s
```

or

```
nm --print-armap
```

to list this index.

An archive with such an index speeds up linking to the library, and allows routines in the library to call each other without regard to their placement in the archive.

The GNU `ranlib` program is another form of GNU `ar`; running `ranlib` is completely equivalent to executing '`ar -s`'.

OPTIONS

`-v` Print the version number of `ranlib` and exit.

SEE ALSO

Utility Programs - `ar`, `nm`

Documents - *The GNU Binary Utilities*, Roland H. Pesch.

COPYING

NAME

`rarpd` – TCP/IP Reverse Address Resolution Protocol server

SYNOPSIS

`rarpd` *EthernetDevice* **`[ debug ]`**

DESCRIPTION

The `rarpd` daemon responds to Reverse Address Resolution Protocol (RARP) requests. It uses the Raw Ethernet Interface (see `raw_ethernet(4)`) support to listen for RARP requests and to send responses to them. The `rarpd` daemon forks a copy of itself and runs in the background; it must be started by the root user.

RARP is typically used by diskless machines to find their Internet Protocol (IP) address during boot time. At boot time, a diskless machines makes a RARP request broadcast packet and provides a 6 byte Ethernet address. The `rarpd` daemon reads this request message and checks it with the `/etc/ethers` and `/etc/hosts` files to find matching entries. If there is a matching entry, `rarpd` will make a RARP response packet and send that packet to the client machine. If there is no matching entry, the RARP request will be ignored.

OPTIONS

EthernetDevice	LynxOS driver name of the Ethernet device being used with RARP.
`debug`	causes the `rarpd` daemon to print the request and response packets onto the console

FILES

`/etc/ethers`	Ethernet address to hostname database
`/etc/hosts`	hostname database

SEE ALSO

Utility Programs - `ifconfig`

Device Drivers - `raw_ethernet`

File Formats - `ethers, hosts`

NAME

`rcp` – remote file copy

SYNOPSIS

rcp [-p] *file1 file2*

rcp [-rp] *file ... directory*

DESCRIPTION

`rcp` copies files between machines. Each *file* or *directory* argument is either a remote file name of the form `rhost:path`, or a local file name (containing no : characters, or a / before any :).

If the **-r** is specified and any of the source files are directories, `rcp` copies each subtree rooted at that name; in this case the destination must be a directory.

If the **-p** is specified, the `mtime` and `atime` of the destination will be preserved, that is, set to those of the sources.

If a `path` of a remote file name is not a full path name, it is interpreted relative to the login directory of the invoking user on `rhost`. A `path` on a remote host may be quoted (using \ or quotes) so that the metacharacters are interpreted remotely.

`rcp` does not prompt for passwords. Your current local user name must exist on `rhost` and allow remote command execution via `rsh`.

`rcp` handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form `rhost.rname` to use `rname` rather than the current user name on the remote host.

NOTE

- Doesn't detect all cases where the target of a copy might be a file in cases where only a directory should be legal.
- Is confused by any output generated by commands in any shell startup files on the remote host.

- The **-r** option depends on `cp` in order to work.
- On the remote machine, `rshd` invokes `dlsh` which in turn invokes the `rcp` program. When the `dlsh` program is invoked, it executes the `.dlshrc` file in the user's login directory. The `.dlshrc` file should not have any commands which result in some output being sent to the standard output.

FILES

`/etc/hosts.equiv`	(on remote host).
`$HOME/.rhosts`	(on remote host).

SEE ALSO

Utility Programs - `ftp, rsh, cp`

NAME

`rcs` – change RCS file attributes

SYNOPSIS

rcs [*options* **]** *file* ...

DESCRIPTION

`rcs` creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For `rcs` to work, the caller's login name must be on the access list, except if the access list is empty, the caller is the owner of the file or the superuser, or the **-i** option is present.

Files ending in "`,v`" are RCS files, all others are working files. If a working file is given, `rcs` tries to find the corresponding RCS file first in directory `./RCS` and then in the current directory, as explained in `co`.

OPTIONS

-i Creates and initializes a new RCS file, but does not deposit any revision. If the RCS file has no path prefix, `rcs` tries to place it as follows. If a file named `RCSDIR` in the current directory exists, it is taken to be a list of directory names (one per line), and each of those is tried. If all fail, or if no `RCSDIR` file exists, then `rcs` tries a directory called `./RCS`, and then finally the current directory itself. If the RCS file already exists, an error message is printed.

-a *logins* Appends the login names appearing in the comma-separated list *logins* to the access list of the RCS file.

-A *oldfile* Appends the access list of *oldfile* to the access list of the RCS file.

-e *logins* Erases the login names appearing in the comma-separated list *logins* from the access list of the RCS file. If *logins* is omitted, the entire access list is erased.

-c *string* — Sets the comment leader to *string*. The comment leader is printed before every log message line generated by the keyword `Log` during checkout (see `co`). This is useful for programming languages without multi-line comments. During `rcs` **-i** or initial `ci`, the comment leader is guessed from the suffix of the working file.

-l *rev* — Locks the revision with number *rev*. If a branch is given, the latest revision on that branch is locked. If *rev* is omitted, the latest revision on the trunk is locked. Locking prevents overlapping changes. A lock is removed with `ci` or `rcs` **-u** (see below).

-u *rev* — Unlocks the revision with number *rev*. If a branch is given, the latest revision on that branch is unlocked. If *rev* is omitted, the latest lock held by the caller is removed. Normally, only the locker of a revision may unlock it. Somebody else unlocking a revision breaks the lock. This causes a mail message to be sent to the original locker. The message contains a commentary solicited from the breaker. The commentary is terminated with a line containing a single "."or **Ctrl-D**.

-L — Sets locking to "strict". Strict locking means that the owner of an RCS file is not exempt from locking for checkin. This option should be used for files that are shared.

-U — Sets locking to "non-strict". Non-strict locking means that the owner of a file need not lock a revision for checkin. This option should NOT be used for files that are shared. The default (**-L** or **-U**) is determined by your system administrator.

-n *name*:*rev* — Associates the symbolic name *name* with the branch or revision *rev*. `rcs` prints an error message if *name* is already associated with another number. If *rev* is omitted, the symbolic *name* is deleted.

-N *name*:*rev* — Same as **-n**, except that it overrides a previous assignment of *name*.

-o *range* Deletes (outdates) the revisions given by *range*. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form *rev1*-*rev2* means revisions *rev1* to *rev2* on the same branch; -*rev* means from the beginning of the branch containing *rev* up to and including *rev*; and *rev*-means from revision *rev* to the end of the branch containing *rev*. None of the outdated revisions may have branches or locks.

-q Quiet mode; diagnostics are not printed.

-s *state*:*rev* Sets the state attribute of the revision *rev* to *state*. If *rev* is omitted, the latest revision on the trunk is assumed; If *rev* is a branch number, the latest revision on that branch is assumed. Any identifier is acceptable for *state*. A useful set of states is "Exp" (for experimental), "Stab" (for stable), and "Rel" (for released). By default, `ci` sets the state of a revision to "Exp".

-t *txtfile* Writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, `rcs` prompts the user for text supplied from the standard input, terminated with a line containing a single "." or control-D. Otherwise, the descriptive text is copied from the file *txtfile*. If the **-i** option is present, descriptive text is requested even if **-t** is not given. The prompt is suppressed if the standard input is not a terminal.

DIAGNOSTICS

The RCS file name and the revisions outdated are written to the diagnostic output. The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 otherwise.

The caller of the command must have read/write permission for the directory containing the RCS file and read permission for the RCS file itself. `rcs` creates a semaphore file in the same directory as the RCS file to prevent simultaneous update. For changes, `rcs` always creates a new file. On successful completion, `rcs` deletes the old one and renames the new one. This strategy makes links to RCS files useless.

COPYRIGHT NOTICE

The RCS system is the work of Walter F. Tichy of Purdue University. It was obtained by Lynx from the Free Software Foundation. As such, Lynx Real-Time Systems will make available for nominal postage and materials cost complete source code for all parts of the RCS system. Furthermore, LynxOS users are hereby permitted to copy and distribute RCS at will under these same terms; that is, notice that source is available from Lynx Real-Time Systems must be given along with any RCS distribution, or else the source itself. If any RCS source is to be used as a component of some other software system, the Free Software Foundation should be contacted for details:

Free Software Foundation, Inc.
675 Massachusetts Ave.
Cambridge, MA 02139

Note that RCS as distributed by FSF has been modified in small ways by Lynx Real-Time Systems, Inc., and thus although source for RCS may be available elsewhere it will be different from LynxOS RCS. Lynx Real-Time Systems assumes no responsibility for copies of RCS obtained elsewhere.

SEE ALSO

Utility Programs - `ci, co, ident, rcsdiff, rcsintro, rlog`

File Formats - `rcsfile`

NAME

`rcsdiff` – compare RCS revisions

SYNOPSIS

rcsdiff [-biwt] [-cefhn] [-r *rev1* **] [-r** *rev2* **] file ...**

DESCRIPTION

`rcsdiff` runs `diff` to compare two revisions of each `RCS` file given. A file name ending in "`,v`" is an `RCS` file name, otherwise a working file name. `rcsdiff` derives the working file name from the `RCS` file name and vice versa, as explained in `co`. Pairs consisting of both an `RCS` and a working file name may also be specified.

All options except **-r** have the same effect as described for `diff`.

If both *rev1* and *rev2* are omitted, `rcsdiff` compares the latest revision on the trunk with the contents of the corresponding working file. This is useful for determining what you changed since the last checkin.

If *rev1* is given, but *rev2* is omitted, `rcsdiff` compares revision *rev1* of the `RCS` file with the contents of the corresponding working file.

If both *rev1* and *rev2* are given, `rcsdiff` compares revisions *rev1* and *rev2* of the `RCS` file.

Both *rev1* and *rev2* may be given numerically or symbolically.

EXAMPLES

The command

```
rcsdiff f.c
```

runs `diff` on the latest trunk revision of `RCS` file `f.c,v` and the contents of working file `f.c`.

COPYRIGHT NOTICE

SEE ALSO

Utility Programs - `ci, co, ident, rcs, rcsintro, rlog`

File Formats - `rcsfile`

NAME

rcsintro – introduction to RCS commands

DESCRIPTION

The Revision Control System (RCS) manages multiple revisions of text files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently; e.g. programs, documentation, graphics, papers, form letters, etc.

The basic user interface is extremely simple. The novice only needs to learn two commands: `ci` and `co`. `ci`, short for “checkin”, deposits the contents of a text file into an archival file called an RCS file. An RCS file contains all revisions of a particular text file. `co`, short for “checkout”, retrieves revisions from an RCS file.

COPYRIGHT NOTICE

The RCS system is the work of Walter F. Tichy of Purdue University. It was obtained by Lynx from the Free Software Foundation. As such, Lynx Real-Time Systems will make available for nominal postage and materials cost complete source code for all parts of the RCS system. Furthermore, LynxOS users are hereby permitted to copy and distribute RCS at will under these same terms; that is, notice that source is available from Lynx Real-Time Systems must be given along with any RCS distribution, or else the source itself. If any RCS source is to be used as a component of some other software system, the Free Software Foundation should be contacted for details:

Free Software Foundation, Inc.
675 Massachusetts Ave.
Cambridge, MA 02139

Note that RCS as distributed by Free Software Foundation has been modified in small ways by Lynx Real-Time Systems, Inc.; thus, although source for RCS may be available elsewhere, it will be different from LynxOS RCS. Lynx Real-Time Systems assumes no responsibility for copies of RCS obtained elsewhere.

SEE ALSO

Utility Programs - `ci, co, ident, rcs, rcsdiff, rlog`

File Formats - `rcsfile`

NAME

`reboot` – reboot the system

SYNOPSIS

`reboot [ -adfhmnprsN ]`

DESCRIPTION

`reboot` calls the operating system `reboot()` function to restart the entire system.

OPTIONS

The options to `reboot` allow bits to be set in the flag word passed to the operating system:

Option	Flag Word	Description
-	`NULL`	Boot system to single user mode.
-a	`RB_AUTOBOOT`	Boots system to multi-user mode. Non-interactive boot from default bootstrap.
-d	`RB_NOSYNC`	Don't sync before reboot.
-f	`RB_NOFSCK`	Check file systems after boot.
-h	`RB_HALT`	Halt the processor for manual reboot.
-m	`RB_MFSCK`	Simulate corrupted disk cache upon reboot.
-n	`RB_ASKNAME`	Boot monitor will request bootstrap name.
-p	`RB_PANIC`	Simulate OS panic.
-r	`RB_RRONLY`	Mount the root file system in read-only mode.
-s	`RB_NOTREALLY`	Set reboot flags, but don't really reboot.
-N	`RB_MAKENODES`	Instructs `init` to create nodes in `/dev` starting `rc`

EXAMPLES

The most common type of `reboot` is:

`reboot -a`

This command sequence reboots LynxOS to multi-user mode after syncing the disks. It also skips the file system checks performed by the `rc` command file (see `init(1)` for more details).

To boot LynxOS in multi-user mode after performing a file systems check, use the **`-m`** option to force the file systems check together with the **`-a`** option.

`reboot -am`

To reboot LynxOS to single user mode, use the **`-`** option as shown below:

`reboot -`

If you need to test a new operating system configuration, invoke `reboot` with the following options:

`reboot -anN`

These options sync the disks and return to the low-level monitor. The low-level monitor requests the name of the new bootstrap program (the newly configured OS). The **`-N`** switch, specifying that device nodes will be rebuilt, must be used if the new LynxOS kernel contains new device drivers and devices.

If you invoke reboot with no options:

`reboot`

then reboot will use the options from the previous reboot. For example, if the system was previously booted with `reboot -a` then issuing `reboot` with no options will result in the system booting to multi-user mode.

NOTES

Only user 0 (root) has permission to use `reboot`.

SEE ALSO

Utility Programs - `fsck, init, sync`

System Calls - `reboot(), sync()`

NAME

`rexecd` – remote execution server

SYNOPSIS

`/etc/rexecd`

DESCRIPTION

`rexecd` is the server for the `rexec` routine. The server provides remote execution facilities with authentication based on user names and passwords.

`rexecd` listens for service requests at the port indicated in the `exec` service specification; see `services`. When a service request is received the following protocol is initiated:

1. The server reads characters from the socket up to a null byte. The resultant string is interpreted as an ASCII number, base 10.
2. If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the `stderr`. A second connection is then created to the specified port on the client's machine.
3. A null-terminated user name of at most 16 characters is retrieved on the initial socket.
4. A null-terminated, unencrypted password of at most 16 characters is retrieved on the initial socket.
5. A null-terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.
6. `rexecd` then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.

7. A null byte is returned on the initial socket and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by `rexecd`.

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the initial socket, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

`Username too long`
: The name is longer than 16 characters.

`Password too long`
: The password is longer than 16 characters.

`Command too long`
: The command line passed exceeds the size of the argument list (as configured into the system).

`Login incorrect`
: No password file entry for the user name existed.

`Password incorrect`
: The wrong password was supplied.

`No remote directory`
: The `pfchdir` command to the home directory failed.

`Try again`
: A fork by the server failed.

`<shellname>: ...`
: The user's login shell could not be started. This message is returned on the connection associated with the `stderr`, and is not preceded by a flag byte.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `rexec`

NAME

rlog – print RCS log information

SYNOPSIS

rlog [*options* **]** *file ...*

DESCRIPTION

rlog prints information about RCS files. Files ending in “**,**v” are RCS files, all others are working files. If a working file is given, rlog tries to find the corresponding RCS file as explained in the description of the co command. rlog by default prints the following information for each RCS file: RCS file name, working file name, head (i.e., the number of the latest revision on the trunk), access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, rlog prints revision number, author, date/time, state, number of lines added/deleted (with respect to the previous revision), locker of the revision (if any), and log message. The options below restrict this output.

OPTIONS

–L Ignores RCS files that have no locks set; convenient in combination with **–b**, **–R**, **–h**, or **–l**.

–R Only prints the name of the RCS file; convenient for translating a working file name into an RCS file name.

–h Prints only RCS file name, working file name, head, access list, locks, symbolic names, and suffix.

–t Prints the same as **–h**, plus the descriptive text.

–b *format* Prints information from the header according to the given *format*. The format string may contain arbitrary text, which is simply printed, intermixed with special markers to select elements from the header:

%**r** Selects the RCS file name.

%**w** Selects the working file name.

`%`**`h`**	Selects the "head" (most recent) revision number.
`%`**`l`**	Selects the list of locks on the file. Each lock is printed as `user:rev`, where user is the `user` name of the locker, and `rev` is the revision number locked. Each pair is separated from the next by a space.
`%`**`s`**	Selects the list of symbolic names associated with the file. Each symbols is shown as `symbol:rev`, where `rev` is the revision number associated with the `symbol`. Each pair is separated from the next by a space.
`%S`(*sym*)	Produces the revision number associated with symbol *sym*, if any.
`%`**`n`**	Instructs `rlog` not to append a newline to the string produced from the format. Newlines included explicitly in the format will be printed.

If no format is given, `rlog` uses the string `%w (%r) locks: %l`.

`-d` *dates*	Prints information about revisions with a checkin date/time in the ranges given by the semicolon-separated list of *dates*. A range of the form *d1*< *d2* or *d2*>*d1* selects the revisions that were deposited between *d1* and *d2*, (inclusive). A range of the form `<d` or `d>` selects all revisions dated `d` or earlier. A range of the form `d<` or `>d` selects all revisions dated `d` or later. A range of the form `d` selects the single, latest revision dated `d` or earlier. The date/time strings `d`, *d1*, and *d2* are in the free format explained in `co`. Quoting is normally necessary, especially for < and >. Note that the separator is a semicolon.
`-l` *lockers*	Prints information about locked revisions. If the comma-separated list *lockers* of login names is given, only the revisions locked by the given login names are printed. If the list is omitted, all locked revisions are printed.
`-r` *revisions*	Prints information about revisions given in the comma-separated list revisions of revisions and ranges. A range *rev1-rev2* means revisions *rev1* to *rev2* on the same branch; *-rev* means revisions from the beginning of the

branch up to and including *rev*; and *rev-* means revisions starting with *rev* to the end of the branch containing *rev*. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range.

-s *states* Prints information about revisions whose state attributes match one of the states given in the comma-separated list *states*.

-w *logins* Prints information about revisions checked in by users with login names appearing in the comma-separated list *logins*. If *logins* is omitted, the user's login is assumed.

`rlog` prints the intersection of the revisions selected with the options **-d**, **-l**, **-s**, **-w**, intersected with the union of the revisions selected by **-b** and **-r**.

EXAMPLES

```
rlog -L -R RCS/*,v
rlog -L -h RCS/*,v
rlog -L -l RCS/*,v
rlog RCS/*,v
rlog -b'%w: %r; co %w' *.c > RCSrules
```

The first command prints the names of all `RCS` files in the subdirectory `RCS` which have locks. The second command prints the headers of those files, and the third prints the headers plus the log messages of the locked revisions. The fourth command prints complete information. The last command produces a file called `RCSrules` containing make rules to allow working `.c` files to be automatically checked out if they do not exist or if the `RCS` file is newer.

DIAGNOSTICS

The exit status always refers to the last `RCS` file operated upon, and is `0` if the operation was successful; `1` otherwise.

COPYRIGHT NOTICE

The RCS system is the work of Walter F. Tichy of Purdue University. It was obtained by Lynx from the Free Software Foundation. As such, Lynx Real-Time Systems will make available for nominal postage and materials cost complete source code for all parts of the RCS system. Furthermore, LynxOS users are hereby permitted to copy and distribute RCS at will under these same terms; that is, notice that source is available from Lynx Real-Time Systems must be given along with any RCS distribution, or else the source itself. If any RCS source is to be used as a component of some other software system, the Free Software Foundation should be contacted for details:

Free Software Foundation, Inc.
675 Massachusetts Ave.
Cambridge, MA 02139

Note that RCS as distributed by FSF has been modified in small ways by Lynx Real-Time Systems, Inc., and thus although source for RCS may be available elsewhere it will be different from LynxOS RCS. Lynx Real-Time Systems assumes no responsibility for copies of RCS obtained elsewhere.

SEE ALSO

Utility Programs - `ci, co, ident, rcs, rcsdiff, rcsintro`

File Formats - `rcsfile`

NAME

`rlogin` – remote login

SYNOPSIS

rlogin *rhost* **[-e***c* **] [-l** *username***] [-8] [-L]**

DESCRIPTION

`rlogin` effectively connects a terminal on the current local host system to the remote host system `rhost`.

Each host has a list of `rhosts` in the file `/etc/hosts.equiv` with which it shares account names. The host names must be the standard names as described in `rsh`. When a user requests `rlogin` as the same user on an equivalent host, the password for that account need not be given. Each user may also have a private equivalence list in the file `.rhosts` in his login directory. Each line in this file should contain an `rhost` and a `username` separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in `login`. To avoid some security problems, the `.rhosts` file must be owned by either the remote user or root, and may not be a symbolic link.

The remote terminal type is the same as the local terminal type (as given in your environment `TERM` variable). All echoing takes place at the remote site, so that (except for delays) the connection is transparent. Flow control via `^S` and `^Q` (DC1 and DC3) and flushing of input and output on interrupts are handled properly. A line of the form **~.** disconnects from the remote host, where ~ is the escape character.

OPTIONS

-e*c*	Specify a different escape character (c) for the line used to disconnect from the remote host.
-l *username*	Specify a different *username* for the remote login. If this option is not used, the remote *username* will be the same as the local *username*.

-8	Pass 8-bit data across the net instead of 7-bit data.
-L	Allow `rlogin` session to run in "litout" mode.

FILES

`/usr/hosts`	For `rhost` version of the command.
`/etc/hosts.equiv`	
`$HOME/.rhosts`	
`/etc/inetd.conf`	Current `inetd` configuration file.

NOTE

More terminal characteristics should be propagated.

SEE ALSO

Utility Programs - `inetd, rcp, rsh, login, telnet`

File Formats - `inetd.conf, hosts.equiv, rhosts`

NAME

`rlogind` – remote login server

SYNOPSIS

`rlogind [-ln ]`

DESCRIPTION

`rlogind` is the server for the `rlogin` program. The server provides a remote login facility with authentication based on privileged port numbers from trusted hosts.

`rlogind` listens for service requests at the port indicated in the `login` service specification; see `services`. When a service request is received, the following protocol is initiated:

1. The server checks the client's source port. If the port is not in the range `512-1023`, the server aborts the connection.
2. The server checks the client's source address and requests the corresponding host name (see `gethostent`, `hosts`, and `named`). If the hostname cannot be determined, the dot-notation representation of the host address is used.

Once the source port and address have been checked, `rlogind` proceeds with the authentication process described in `rshd`. It then allocates a pseudo terminal (see `pty`), and manipulates file descriptors so that the slave half of the pseudo terminal becomes the `stdin`, `stdout`, and `stderr` for a login process. The login process is an instance of the `login` program, invoked with the **`-f`** option if authentication has succeeded. If automatic authentication fails, the user is prompted to log in as if on a standard terminal line. The **`-l`** option prevents any authentication based on the user's `.rhosts` file, unless the user is logging in as the superuser.

The parent of the login process manipulates the master side of the pseudo terminal, operating as an intermediary between the login process and the client instance of the `rlogin` program. In normal operation, the packet protocol described in `pty` is invoked to provide `^S/^Q` type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, `TERM` (see `environ`). The screen or window size of the terminal is requested from the client, and window size changes from the client are propagated to the pseudo terminal.

Transport-level "keep alive" messages are enabled unless the **-n** option is present. The use of keep alive messages allows sessions to be timed out if the client crashes or becomes unreachable.

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the `stderr`, after which any network connections are closed. An error is indicated by a leading byte with a value of `1`.

`Tryagain.`	A fork by the server failed.
`/bin/sh: ...`	The user's login shell could not be started.

NOTE

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

A more extensible protocol should be used.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs -	`rshd, inetd`
File Formats -	`inetd.conf`

NAME

`rm, rmdir` – remove (unlink) files or directories

SYNOPSIS

rm [*options***]** *file ...*

rmdir *directory ...*

DESCRIPTION

`rm` removes directory entries for files. If an entry was the last link to a file, the file is removed as well. Removal of an entry requires write permission in its directory. If a file is not writable, and `rm` is using the current terminal for its standard input, the file permissions are printed on the screen and a response is read from the terminal. `rm` will proceed to remove the file if the response is `y`.

`rmdir` removes directories. Each *directory* must be empty except for the special entries "`.`" and "`..`".

OPTIONS

–	Allow all arguments following to be treated as filenames. This option is useful for filenames which begin with the "–" character
–f	Force removal of unwritable files.
–i	Ask user's permission every time a file or directory is to be removed. If the user responds with a `y`, the file is removed.
–r	Recursively remove a directory subtree. `rm` will not remove a directory unless this option is specified.
–v	Generates verbose output.

SEE ALSO

Utility Programs - `ln, mv`

System Calls - `rmdir(), unlink()`

NAME

`route` – manually manipulate the routing tables.

SYNOPSIS

`route [-nqv]` *command* **`[[`***modifiers***`]`** *args***`]`**

DESCRIPTION

`route` is a utility used to manually manipulate the network routing tables. It normally is not needed, as a system routing table management daemon such as `routed`(1), should tend to this task.

The `route` utility supports a limited number of general options, but a rich command language, enabling the user to specify any arbitrary request that could be delivered via the programmatic interface.

OPTIONS

`-n`	Bypasses attempts to print host and network names symbolically when reporting actions. (The process of translating between symbolic names and numerical equivalents can be quite time consuming, and may require correct operation of the network; thus it may be expedient to forgo this, especially when attempting to repair networking operations),
`-v`	(verbose) Print additional details.
`-q`	Suppress all output.

USAGE

The route: utility provides six commands:

`add`	Add a route.
`flush`	Remove all routes.
`delete`	Delete a specific route.
`change`	Change aspects of a route (such as its gateway).

`get`	Lookup and display the route for a destination.
`monitor`	Continuously report any changes to the routing information base, routing lookup misses, or suspected network partitionings.

The `monitor` command has the syntax

`route [-n] monitor`

The flush command has the syntax

`route [-n] flush [`*family***`]`**

If the `flush` command is specified, `route` will "flush" the routing tables of all gateway entries. When the address family may is specified by any of the **`-osi`**, **`-xns`**, or **`-inet`** modifiers, only routes having destinations with addresses in the delineated family will be deleted.

The other commands have the following syntax:

`route [-n]` *command* **`[-net|-host]`** *destination gateway*

where *destination* is the destination host or network, *gateway* is the next-hop intermediary via which packets should be routed. Routes to a particular host may be distinguished from those to a network by interpreting the Internet address specified as the *destination* argument. The optional modifiers **`-net`** and **`-host`** force the destination to be interpreted as a network or a host, respectively. Otherwise, if the *destination* has a "local address part" of `INADDR_ANY` , or if the *destination* is the symbolic name of a network, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host.

For example,

```
128.32
```

is interpreted as

```
-host 128.0.0.32
```

and

```
128.32.130
```

is interpreted as

```
-host 128.32.0.130
```

and

```
-net 128.32
```

is interpreted as

```
128.32.0.0
```

and

```
-net 128.32.130
```

is interpreted as

```
128.32.130.0
```

If the destination is directly reachable via an interface requiring no intermediary system to act as a gateway, the **-interface** modifier should be specified; the gateway given is the address of this host on the common network, indicating the interface to be used for transmission.

The optional **-netmask** qualifier is intended to achieve the effect of an OSI ESIS redirect with the netmask option, or to manually add subnet routes with netmasks different from that of the implied network interface (as would otherwise be communicated using the OSPF or ISIS routing protocols). One specifies an additional ensuing address parameter (to be interpreted as a network mask). The implicit network mask generated in the AF_INET case can be overridden by making sure this option follows the destination parameter.

Routes have associated flags which influence operation of the protocols when sending to destinations matched by the routes. These flags may be set (or sometimes cleared) by indicating the following corresponding modifiers:

```
-cloning    RTF_CLONING    - generates a new route on use
-xresolve   RTF_XRESOLVE   - emit mesg on use (for external lookup)
-iface     ~RTF_GATEWAY    - destination is directly reachable
-static     RTF_STATIC     - manually added route
-nostatic   RTF_STATIC     - pretend route added by kernel or daemon
-reject     RTF_STATIC     - emit an ICMP unreachable when matched
-blackhole  RTF_BLACKHOLE  - silently discard pkts (during updates)
-proto1     RTF_BLACKHOLE  - set protocol specific routing flag #1
-proto2     RTF_PROTO2     - set protocol specific routing flag #2
-proto2     RTF_PROTO2     - validly translates proto to link addr
```

The optional modifiers **-rtt, -rttvar, -sendpipe, -recvpipe, -mtu,** -hopcount, **-expire,** and **-ssthresh** provide initial values to quantities maintained in the routing entry by transport level protocols, such as TCP or TP4. These may be individually locked by preceding each such modifier to be locked by the **-lock** meta-modifier, or one can specify that all ensuing metrics may be locked by the **-lockrest** meta-modifier.

In a change or add command where the destination and gateway are not sufficient to specify the route (as in the ISO case where several interfaces may have the same address), the **-ifp** or **-ifa** modifiers may be used to determine the interface or interface address.

All symbolic names specified for a *destination* or *gateway* are looked up first as a host name using gethostbyname(3). If this lookup fails, getnetbyname(3) is then used to interpret the name as that of a network.

route uses a routing socket and the new message types RTM_ADD, RTM_DELETE, RTM_GET, and RTM_GET. As such, only the super-user may modify the routing tables.

DIAGNOSTICS

add [host|network] %s: *gateway* %s: *flags*

The specified route is being added to the tables. The values printed are from the routing table entry supplied in the `ioctl`(2) call. If the gateway address used was not the primary address of the gateway (the first one returned by `gethostbyname`(3)), the gateway address is printed numerically as well as symbolically.

`delete [host &| network]%s:` *gateway* `%s` *flags*
: As above, but when deleting an entry.

`%s %s done`
: When the flush command is specified, each routing table entry deleted is indicated with a message of this form.

`Network is unreachable`
: An attempt to add a route failed because the gateway listed was not on a directly-connected network. The next-hop gateway must be given.

`not in table`
: A delete operation was attempted for an entry which wasn't present in the tables.

`routing table overflow`
: An add operation was attempted, but the system was low on resources and was unable to allocate memory to create the new entry.

SEE ALSO

Utility Programs - `routed`

HISTORY

The route command appeared in 4.2BSD.

BUGS

The first paragraph may have slightly exaggerated `routed`'s abilities.

NAME

`routed` – network routing daemon

SYNOPSIS

routed [-d] [-g] [-q] [-s] [-t] [*logfile***]**

DESCRIPTION

`routed` is invoked at boot time to manage the network routing tables. The routing daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining up to date kernel routing table entries. It used a generalized protocol capable of use with multiple address types, but is currently used only for Internet routing within a cluster of networks.

In normal operation `routed` listens on the `udp` socket for the `route` service (see `services`(5)) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When `routed` is started, it uses the `SIOCGIFCONF ioctl` to find those directly connected interfaces configured into the system and marked "up" (the software loopback interface is ignored). If multiple interfaces are present, it is assumed that the host will forward packets between networks. `routed` then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, `routed` formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a "hop count" metric (a count of 16, or greater, is considered "infinite"). The metric associated with each route returned provides a metric *relative to the sender*.

Response packets received by `routed` are used to update the routing tables if one of the following conditions is satisfied:

- No routing table entry exists for the destination network or host, and the metric indicates the destination is "reachable" (i.e. the hop count is not infinite).

- The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.
- The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.
- The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, `routed` records the change in its internal tables and updates the kernel routing table. The change is reflected in the next response packet sent.

In addition to processing incoming packets, `routed` also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the local internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks. The response is sent to the broadcast address on nets capable of that function, to the destination address on point-to-point links, and to the router's own address on other networks. The normal routing tables are bypassed when sending gratuitous responses. The reception of responses on each network is used to determine that the network and interface are functioning correctly. If no response is received on an interface, another route may be chosen to route around the interface, or the route may be dropped if no alternative is available.

OPTIONS

`-d` Enable additional debugging information to be logged, such as bad packets received.

-g This flag is used on internetwork routers to offer a route to the "default" destination. This is typically used on a gateway to the Internet, or on a gateway that uses another routing protocol whose routes are not reported to other local routers.

-s Supplying this option forces `routed` to supply routing information whether it is acting as an internetwork router or not. This is the default if multiple network interfaces are present, or if a point-to-point link is in use.

-q This is the opposite of the **-s** option.

-t If the **-t** option is specified, all packets sent or received are printed on the standard output. In addition, `routed` will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process.

Any other argument supplied is interpreted as the name of file in which the actions of `routed` should be logged. This log contains information about any changes to the routing tables and, if not tracing all packets, a history of recent messages sent and received which are related to the changed route.

USAGE

In addition to the facilities described preceding, `routed` supports the notion of "distant" *passive* and *active* gateways. When `routed` is started up, it reads the file `/etc/gateways` to find gateways which may not be located using only information from the `SIOGIFCONF ioctl`. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (i.e. they should have a `routed` process running on the machine). Routes through passive gateways are in stalled in the kernel's routing tables once upon startup. Such routes are not included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of time, the associated route is deleted. Gateways marked external are also passive, but are not placed in the kernel routing table nor are they included in routing updates. The function of external entries is to inform `routed` that another routing process will install such a route, and that alternate routes to that destination should not be installed. Such entries are only required when both routers may learn of routes to the same destination.

The `/etc/gateways` file is comprised of a series of lines, each in the following format:

`net|host` *dest* `gateway` *fwd* `metric` *hops* `passive|active|external`

The `net` or `host` keyword indicates if the route is to a network or specific host.

dest is the name of the destination network or host. This may be a symbolic name located in `/etc/networks` or `/etc/hosts` (or, if started after `named`, known to the name server), or an Internet address specified in “dot” notation; see `inet_ntoa()` within `inet_addr`(3).

fwd is the name or address of the gateway to which messages should be forwarded.

hops is a metric indicating the hop count to the destination host or network.

One of the keywords `passive`, `active`, or `external` indicates if the gateway should be treated as *passive* or *active* (as described above), or whether the gateway is *external* to the scope of the `routed` protocol.

Internetwork routers that are directly attached to the Arpanet or Milnet should use the Exterior Gateway Protocol (EGP) to gather routing information rather then using a static routing table of passive gateways. EGP is required in order to provide routes for local networks to the rest of the Internet system.

FILES

`/etc/gateways`	for distant gateways

SEE ALSO

Device Facilities -	`udp`, `icmp`
File Formats -	`services`
Documents -	*Internet Transport Protocols*, XSIS 028112, Xerox System Integration Standard.

BUGS

The kernel's routing tables may not correspond to those of `routed` when redirects change or add routes. `routed` should note any redirects received by reading the ICMP packets received via a raw socket.

`routed` should listen to intelligent interfaces, such as an IMP, to gather more information. It does not always detect unidirectional failures in network interfaces (e.g., when the output side fails).

HISTORY

The `routed` command appeared in 4.2BSD.

NOTES

If you are running `routed` from either of the nodes in the SLIP network, make sure to run the `routed` daemon from the other node as well. This ensures that the `routed` daemons can refresh the SLIP route information periodically. If `routed` is started on only one of the SLIP nodes, the `slip` routing entries within the kernel are subjected to the 180 sec timeout; in the absence of periodic routing information from the other node, the slip routing entries will be removed.

Do not run `routed` from any of the nodes configured in an SCMP network. The VME backplane driver used for SCMP doesn't support broadcasting; hence the `routed` daemons are not able to exchange the route information over the backplane. This results in the timeout/removal of the backplane routing entries from the kernel by the routed daemon.

NAME

`rpcgen` – an RPC protocol compiler

SYNOPSIS

rpcgen *infile*

rpcgen -c | -h | -l | -m [-o *outfile* **] [** *infile* **]**

rpcgen -s *transport* **[-o** *outfile* **] [** *infile* **]**

DESCRIPTION

`rpcgen` is a tool that generates C code to implement an RPC protocol. The input to `rpcgen` is a language similar to C known as RPC Language (Remote Procedure Call Language). Information about the syntax of RPC Language is available in the `rpcgen` Programming Guide.

`rpcgen` is normally used as in the first synopsis where it takes an input file and generates four output files. If the infile is named `proto.x`, then `rpcgen` will generate a header file in `proto.h`, XDR routines in `proto_xdr.c`, server-side stubs in `proto_svc.c`, and client-side stubs in `proto_clnt.c`.

The other synopses shown above are used when one does not want to generate all the output files, but only a particular one. Their usage is described in the OPTIONS section below.

The C-preprocessor, `cpp`, is run on all input files before they are actually interpreted by `rpcgen`, so all the `cpp` directives are legal within an `rpcgen` input file. For each type of output file, `rpcgen` defines a special `cpp` symbol for use by the `rpcgen` programmer:

`RPC_HDR`	Defined when compiling into header files.
`RPC_XDR`	Defined when compiling into XDR routines.
`RPC_SVC`	Defined when compiling into server-side stubs.
`RPC_CLNT`	Defined when compiling into client-side stubs.

In addition, `rpcgen` does a little preprocessing of its own. Any line beginning with `%` is passed directly into the output file, uninterpreted by `rpcgen`.

You can customize some of your XDR routines by leaving those data types undefined. For every data type that is undefined, `rpcgen` will assume that there exists a routine with the name `xdr_` prepended to the name of the undefined type.

OPTIONS

-c	Compile into XDR routines.
-h	Compile into C data-definitions (a header file).
-l	Compile into client-side stubs.
-m	Compile into server-side stubs, but do not generate a "main" routine. This option is useful for doing callback-routines and for people who need to write their own "main" routine to do initialization.
-o *outfile*	Specify the name of the output file. If none is specified, standard output is used (**-c** , **-h** , **-l** and **-s** modes only).
-s *transport*	Compile into server-side stubs, using the given transport. The supported transports are `udp` and `tcp`. This option may be invoked more than once so as to compile a server that serves multiple transports.

NOTE

Nesting is not supported. As a work-around, structures can be declared at top-level, and their name used inside other structures in order to achieve the same effect.

Name clashes can occur when using program definitions, since the apparent scope does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures and types.

`rpcgen` allows the user to have a user-defined symbolic link to the C preprocessor. `rpcgen` looks for a user-defined symbolic link in /bin called `rpcgencpp`. If this is not found, it looks in `/usr/local/`lib for `gcc-cpp`.

SEE ALSO

Utility Programs - `cpp`

Documents - *rpcgen Programming Guide*

NAME

`rpcinfo` – report `RPC` information

SYNOPSIS

`rpcinfo -p [` *host* **`]`**
`rpcinfo [ -n` *portnum* **`] -u`** *host program* **`[`** *version* **`]`**
`rpcinfo [ -n` *portnum* **`] -t`** *host program* **`[`** *version* **`]`**
`rpcinfo -b` *program version*
`rpcinfo -d` *program version*

DESCRIPTION

`rpcinfo` makes an `RPC` call to an `RPC` server and reports what it finds.

OPTIONS

`-p` Probe the portmapper on host, and print a list of all registered `RPC` programs. If *host* is not specified, it defaults to the value returned by `hostname`.

`-u` Make an `RPC` call to procedure 0 of *program* on the specified *host* using UDP, and report whether a response was received.

`-t` Make an `RPC` call to procedure 0 of *program* on the specified *host* using TCP, and report whether a response was received.

`-n` Use `portnum` as the port number for the **`-t`** and **`-u`** options instead of the port number given by the portmapper.

`-b` Make an `RPC` broadcast to procedure 0 of the specified *program* and *version* using UDP and report all hosts that respond.

`-d` Delete registration for the `RPC` service of the specified *program* and *version*. This option can be exercised only by the super-user.

The program argument can be either a name or a number.

If a version is specified, `rpcinfo` attempts to call that version of the specified program. Otherwise, `rpcinfo` attempts to find all the registered version numbers for the specified program by calling version 0 (which is presumed not to exist; if it does exist, `rpcinfo` attempts to obtain this information by calling an extremely high version number instead) and attempts to call each registered version. Note: the version number is required for **-b** and **-d** options.

EXAMPLES

To show all of the RPC services registered on the local machine use:

```
rpcinfo -p
```

To show all of the RPC services registered on the machine named `lynxbsd` use:

```
rpcinfo -p lynxbsd
```

To show all machines on the local net that are running the NFS server service use:

```
rpcinfo -b nfs 'version'
```

Where `version` is the current NFS server version obtained from the results of the **-p** switch above.

SEE ALSO

Utility Program - `portmap`

Library Function - `rpc()`

Documentation - *RPC Programming Guide*

NAME

`rpc.lockd.svc`,
`rpc.lockd.clnt` – network lock daemons

SYNOPSIS

/net/rpc.lockd.svc [-g *graceperiod* **]**

/net/rpc.lockd.clnt [*timeout* **] [** *retry-count* **]**

DESCRIPTION

`lockd` processes lock requests that are either sent locally by the kernel or remotely by another lock daemon. `lockd` forwards lock requests for remote data to the server site's lock daemon. `lockd` then requests the status monitor daemon, `rpc.statd`, for monitor service. The reply to the lock request will not be sent to the kernel until the status daemon and the server site's lock daemon have replied.

If either the status monitor or server site's lock daemon is unavailable, the reply to a lock request for remote data is delayed until all daemons become available or the timeout has expired.

When a server recovers, it waits for a grace period for all client site lock daemons to submit reclaim requests. Client site lock daemons, on the other hand, are notified by the status daemon of the server recovery and promptly resubmit previously granted lock requests. If `lockd` fails to secure a previously granted lock at the server site, it sends `SIGLOST` to a process.

OPTIONS

timeout	Use *timeout* (seconds) as the interval instead of the default value (15 seconds) to retransmit lock request to the remote server.
graceperiod	Use *graceperiod* (seconds) as the grace period duration instead of the default value (45 seconds).

SEE ALSO

Utility Programs-	`rpc.statd`
System Calls-	`fcntl()`

NAME

`rpc.statd` – network status monitor

SYNOPSIS

`/net/rpc.statd`

DESCRIPTION

`rpc.statd` is an intermediate version of the status monitor. It interacts with `rpc.lockd` to provide the crash and recovery functions for the locking services on NFS.

FILES

`/etc/sm`

`/etc/state`

BUGS

The crash of a site is only detected upon its recovery.

NAME

`rsh` – remote shell

SYNOPSIS

`rsh host` [**`-l`** *username*] [**`-n`**] *command* **`host`** [**`-l`** *username*] [**`-n`**] *command*

DESCRIPTION

`rsh` connects to the specified host, and there executes the specified command. `rsh` copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit, and terminate signals are propagated to the remote command; `rsh` normally terminates when the remote command does.

The remote username used is the same as the current local username, unless a different remote name is given with the -l option. This remote name must be equivalent to the originating account; no provision is made for specifying a password with a command.

If command is omitted, then instead of executing a single command, `rsh` chains to `rlogin` to log into the remote host.

Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command

```
rsh otherhost cat remotefile >> localfile
```

appends the remote file `remotefile` to the local file `localfile`, while

```
rsh otherhost cat remotefile `>>' otherremotefile
```

appends `remotefile` to `otherremotefile`.

the -n option redirects output to `/dev/null`. This option is useful to avoid unexpected interaction between `rsh` and the shell that invokes the `rsh` command.

Host names are given in the file `/etc/hosts`. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The host names for local machines are sometimes made commands in the directory `/usr/hosts`; if you put this directory in the search path, then the word `rsh` can be omitted.

NOTE

`rlogin` must be located in the standard search path in order to be found by a command-less invocation of `rsh`.

Stop signals stop the local `rsh` process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

FILES

`/etc/hosts`	
`/usr/hosts/*`	
`/etc/hosts.equiv`	(on remote host)
`$HOME/.rhosts`	(on remote host)

SEE ALSO

Utility Programs -	`rlogin, rcp`
File Formats -	`hosts`

NAME

`rshd` – remote shell server

SYNOPSIS

`rshd`

DESCRIPTION

`rshd` is the server for the `rcmd` routine and, consequently, for the `rsh` program. The server provides remote execution facilities with authentication based on privileged port numbers from trusted hosts.

`rshd` listens for service requests at the port indicated in the `cmd` service specification; see services. When a service request is received the following protocol is initiated:

1. The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection.
2. The server reads characters from the socket up to a null (`\0`) byte. The resultant string is interpreted as an ASCII number, base 10.
3. If the number received in step 2 is non-zero, it is interpreted as the port number of a secondary stream to be used for the `stderr`. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 0-1023.
4. The server checks the client's source address and requests the corresponding host name (see `gethostent`, `hosts`, and `named`). If the hostname cannot be determined, the dot-notation representation of the host address is used.
5. A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the client's machine.
6. A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the server's machine.

7. A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

8. `rshd` then validates the user according to the following steps. The local (server-end) user name is looked up in the password file and a `chdir` is performed to the user's home directory. If either the lookup or `chdir` fail, the connection is terminated. If the user is not the super-user, (user ID 0), the file `/etc/hosts.equiv` consulted for a list of hosts considered "equivalent". If the client's host name is present in this file, the authentication is considered successful.

   If the lookup fails, or the user is the super-user, then the file `.rhosts` in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.

9. A null byte is returned on the initial socket and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by `rshd`.

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the initial socket, after which any network connections are closed.

An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the execution of the login shell).

`locuser too long`
: The name of the user on the client's machine is longer than 16 characters.

`remuser too long`
: The name of the user on the remote machine is longer than 16 characters.

`command too long`
: The command line passed exceeds the size of the argument list (as configured into the system).

`Login incorrect.`
No password file entry for the user name existed.

`No remote directory.`
The `chdir` command to the home directory failed.

`Permission denied.`
The authentication procedure described above failed.

`Can't make pipe.`
The pipe needed for the `stderr`, wasn't created.

`Try again.` A `fork` by the server failed.

`<shellname>: ...`
The user's login shell could not be started. This message is returned on the connection associated with the `stderr`, and is not preceded by a flag byte.

NOTE

The authentication procedure used here assumes the integrity of each client machine and the connecting medium.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `rsh`

NAME

`runtest` – the DejaGnu test driver program

SYNOPSIS

`runtest [` *options* **`]`**

DESCRIPTION

DejaGnu is a framework for running test suites on GNU tools. It is written in expect, which uses TCL (Tool command language). `runtest` is the test driver program; use it to control what tests to run, and variations on how to run them.

You can find a comprehensive description of DejaGnu and `runtest` in *The DejaGnu Testing Framework* or its Info version, `dejagnu.info`.

OPTIONS

`--all` Print all test output to screen. By default, only unexpected results are displayed.

`--baud` *rate* Set the baud rate for a serial line connection. Some serial interface programs (like `tip`) don't use this value but instead use a separate initialization file.

`--connect` *type* The type of connection to use. The choices are `rlogin`, `telnet`, `rsh`, `kermit`, `tip` and `mondfe`.

`--debug` Turn on `expect` internal debugging output. All output is logged to a file called `dbg.out`. The output of the **`--strace`** also goes into this file.

`--help` Prints out a help screen and then exits.

`--host` *type* The configuration string for the host.

`--ignore` *test1.exp test2.exp ...*
Do not run the specified tests.

`--mail` *'name1 name2 ...'*
Electronic mail addresses to receive test results.

--name *hostname*
The network hostname of the target board.

--objdir *path* *path* is a directory containing compiled test code.

--outdir *directory*
The name of a directory for test log output.

--reboot Reboot the target board when `runtest` initializes (if supported).

--srcdir *path* *path* is a directory containing test directories.

--strace *N* Turns on `expect` internal tracing to *N* levels deep.

--target *type* The configuration string for the target.

--tool *toolname* Specify the tool to be tested. *toolname* controls the test suite applied, and the associated initialization module.

--verbose, -v Turns on more debugging output from test cases and DejaGnu utility code. Use more than once to increase output further.

--version, -V Prints out the versions of DejaGnu, expect and Tcl.

-D[number] Activate the Tcl debugger.`number` can be either 1 or 0. If it is 1, then the expect shell will break when it starts to run. All ^C's drop DejaGnu back to the debugger prompt. A 0 starts DejaGnu like normal, but a ^C drops to the debugger prompt.

Any file name on the command line is assumed to be a subset of the test names to run. Usually these are the names of the expect test driver, ie... `special.exp`.

Makefile style variables are used to specify tool names and their flags; these and other configuration dependent values are saved in the file `site.exp`, created during configuration.

EXIT CODES

`runtest` sets the exit code to 1 if any of the tests failed, or sets it to 0 if all the tests passed.

SEE ALSO

Documents - *The DejaGnu Testing Framework* (`dejagnu.info`). This is the DejaGnu manual; its source is the Texinfo file `doc/dejagnu.texi` in the DejaGnu distribution.

AUTHOR

Rob Savoye (rob@cygnus.com)

NAME

`ruptime` – show host status of local machines

SYNOPSIS

`ruptime [ -a ] [ -r ] [-l] [ -t] [ -u]`

DESCRIPTION

`ruptime` gives a status line like `uptime` for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 11 minutes are shown as being down.

Users idle an hour or more are not counted unless the **`-a`** flag is given.

Normally, the listing is sorted by host name. The **`-l`**, **`-t`**, and **`-u`** flags specify sorting by load average, uptime, and number of users, respectively. The **`-r`** flag reverses the sort order.

FILES

`/usr/spool/rwho/whod.*` data files

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `rwho`

NAME

rwho – reports who is logged in on local machines

SYNOPSIS

rwho [-a]

DESCRIPTION

The rwho command produces output similar to who, but for all machines on the local network. If no report has been received from a machine for 5 minutes then rwho assumes the machine is down, and does not report users last known to be logged into that machine.

If a user hasn't typed to the system for a minute or more, then rwho reports this idle time. If a user hasn't typed to the system for an hour or more, then the user will be omitted from the output of rwho unless the **-a** flag is given.

FILES

/usr/spool/rwho/whod.* information about other machines

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley

SEE ALSO

Utility Programs - ruptime, rwhod

NAME

rwhod - system status server

SYNOPSIS

rwhod

DESCRIPTION

rwhod is the server which maintains the database used by the rwho and ruptime programs. Its operation is predicated on the ability to broadcast messages on a network.

rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other rwhod server's status messages, validating them, then recording them in a collection of files located in the directory /usr/spool/rwho.

The server transmits and receives messages at the port indicated in the rwho service specification; see services. The messages sent and received, are of the form:

```
struct    outmp  {
  char    out\_line[8];  /* tty name */
  char    out\_name[8];  /* user id */
  long    out\_time;  /* time on */
};
struct          whod  {
  char    wd\_vers;
  char    wd\_type;
  char    wd\_fill[2];
  int     wd\_sendtime;
  int     wd\_recvtime;
  char    wd\_hostname[32];
  int     wd\_loadav[3];
  int     wd\_boottime;
  struct  whoent {
    struct  outmp  we\_utmp;
```

```
        int      we\_idle;
    } wd\_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the w program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission; they are multiplied by 100 for representation in an integer.

The host name included is that returned by the gethostname system call, with any trailing domain name omitted. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the utmp entry for each non-idle terminal line and a value indicating the time in seconds since a character was last received on the terminal line.

Messages received by the rwho server are discarded unless they originated at a rwho server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by rwhod are placed in files named whod.hostname in the directory /usr/spool/rwho. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 3 minutes. rwhod performs an nlist on /lynx.os every 30 minutes to guard against the possibility that this file is not the system image currently operating.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - rwho, ruptime

NAME

`s5fstotar` – convert SysV file system to `tar` archive format

SYNOPSIS

s5fstotar [-v] *special* **[** *file* **]**

DESCRIPTION

`s5fstotar` converts the data from a mountable System V UNIX file system into `tar` archive format. The input is taken from the special file *special*. The output is written to *file* if given, or to the standard output if not.

If the **-v** option is given, and either *file* has been specified or the standard output is not a terminal, then `s5fstotar` will send verbose output to the standard error channel. The verbose output includes the block size used on the file system being read and information about each directory and file processed.

EXAMPLES

To convert the files from a System V mountable floppy in drive `/dev/fd0` into `tar` archive format in the file `archive`:

```
s5fstotar /dev/fd0 archive
```

To extract the files from a System V mountable floppy in drive `/dev/fd0`:

```
s5fstotar /dev/fd0 | tar xf -
```

In the above example, it is advisable not to use the verbose option of `s5fstotar` in conjunction with the verbose option of `tar` because the verbose output would be redundant. Instead, use the verbose option of one or the other:

```
s5fstotar -v /dev/fd0 | tar xf -
```

or

```
s5fstotar /dev/fd0 | tar xvf -
```

SEE ALSO

Utility Programs - `tar`

File Formats - `tar`

NAME

`scandisk` – test disk drives for errors

SYNOPSIS

scandisk [-b *bsize***] [-d]-[R|B] [-A] [-C] [-r** *block block***]**
[-q] *raw_device*

DESCRIPTION

`scandisk` performs various tests on SCSI or IDE disks and can remap the bad blocks that are found. If a disk has many disk errors, it may be better to reformat it instead of remapping the bad blocks. Bad blocks and the current block being tested will be displayed on the screen.

OPTIONS

-**b** *bsize*	Specifies a logical block size to test. *bsize* must be a multiple of 512 bytes. Larger block sizes scan a disk faster. The maximum value for *bsize* is 65536.
-**r** *block block*	Specifies a minimum and maximum block range to test exclusively. The default range is the whole disk.
-**d**	Destructive testing. This disk will be written to. The default is non-destructive. The user will be prompted and must type **Y** to continue.
-**c**	Continuous testing. `^C` will interrupt testing.
-**R**	Random testing. Random blocks will be tested in the range specified.
-**B**	Butterfly testing. This option will cause data to be read from the first block in the range then the last block in the range. The second block will then be read followed by the second to last block and so on until they get to the opposite side they started on (last block in range followed by first block in range).
-**A**	Auto-remapping. This option will automatically remap bad blocks. This option should not be used with the -**c** option.

-q Quiet mode. Quiet mode is useful for scanning a disk in a non-interactive way. It suppresses all messages except bad block messages. For example:

`bash$ scandisk -q` *device* `> bad1`

The file `bad1` will contain any bad block messages that `scandisk` produces when testing *device*.

NOTE

Disks that are beginning to fail may not respond well to automatic remapping of blocks. A freshly remapped block may still be bad because the new location that represents that block may be bad. If a disk has bad blocks, scan and remap them then re-scan to ensure that the new areas are ok.

The -**q** option with the -**d** option will suppress the user prompt, proceeding with writes to the specified drive.

SEE ALSO

Utility Programs - `fmtscsi`

System Calls - `ioctl()`

NAME

`sed` – stream editor

SYNOPSIS

sed [-n] *script* **[** *file* **]**

sed [-n] [-e *script* **]**...**[-f** *script_file* **]**...**[** *file* ...**]**

DESCRIPTION

The `sed` utility is a stream editor that reads one or more text files, make editing changes according to a script of editing commands, and write the results to standard output. The *script* is obtained from either the script operand string or a combination of the option-arguments from the **-e** *script* and **-f** *script_file* options.

The `sed` utility conforms to the utility argument syntax guidelines described in `getopt()`, except that the order of presentation of the **-e** and **-f** options is significant.

The following options are supported by the implementation:

-e *script* Add the editing commands specified by the script option-argument to the end of the script of editing commands. The *script* option-argument has the same properties as the *script* operand.

-f *script_file* Add the editing commands in the file *script_file* to the end of the script.

-n Suppress the default output (in which each line, after it is examined for editing, is written to standard output). Only lines explicitly selected for output are written.

Multiple **-e** and **-f** options may be specified. All commands are added to the script in the order specified, regardless of their origin.

OPERANDS

The following operands are supported by the implementation:

file A pathname of a file whose contents are read and edited. If multiple *file* operands are specified, the named files are read in the order specified and the concatenation is edited. If no *file* operands are specified, the standard input is used.

script A string to be used as the script of editing commands. The application will not present a *script* that violates the restrictions of a text file, except that the final character need not be a `<newline>`.

EXTERNAL INFLUENCES

Standard Input The standard input is used only if no *file* operands are specified. See Input Files.

Input Files The input files are text files. The *script_files* named by the **-f** option consist of editing commands, one per line.

EXTERNAL EFFECTS

Standard Output The input files are written to standard output, with the editing commands specified in the script applied. If the **-n** option is specified, only those input lines selected by the script are written to standard output.

Standard Error Used only for diagnostic messages.

Output Files The output files are text files whose formats are dependent on the editing commands given.

EXTENDED DESCRIPTION

The script consists of editing commands, one per line, of the following form:

`[address[,address]]`*command*`[arguments]`

Zero or more `<blank>`s are accepted before the first address and before *command*.

In default operation, sed cyclically copies a line of input, less its terminating <newline>, into a *pattern space* (unless there is something left after a D command), applies in sequence all commands whose addresses select that pattern space, and at the end of the script copies the pattern space to standard output (except when **-n** is specified) and deletes the pattern space. Whenever the pattern space is written to standard output or a named file, sed immediately follows it with a <newline>.

Some of the commands use a *hold space* to save all or part of the pattern space for subsequent retrieval. The pattern and hold spaces shall each be able to hold at least 8192 bytes.

sed Addresses

An address is either empty, a decimal number that counts input lines cumulatively across files, a $ character that addresses the last line of input, or a context address (which consists of a regular expression, preceded and followed by a delimiter, usually a slash).

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

a command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line will be selected.) Starting at the first line following the selected range, sed looks again for the first address. Thereafter, the process is repeated.

Editing commands can be applied only to non-selected pattern spaces by use of the negation command **!**.

sed Regular Expressions

The sed utility supports the basic regular expressions, with the following additions:

1. In a context address, the construction \cREc, where c is any character other than backslash or <newline>, is identical to /RE/. If the character designated by c appears following a

backslash, then it is considered to be that literal character, which will not terminate the RE. For example, in the context address `\xabc|\xdefx`, the second `x` stands for itself, so that the regular expression is `absxdef`.

2. The escape sequence `\n` matches a <newline> embedded in the pattern space. A literal <newline> character must not be used in the regular expression of a context address or in the substitute command.

sed Editing Commands

In the following list of commands, the maximum number of permissible addresses for each command is indicated by [*0addr*], [*1addr*], or [*2addr*], representing zero, one, or two addresses.

The argument *text* consists of one or more lines. Each embedded <newline> in the text is preceded by a backslash. Other backslashes in test are removed and the following character is treated literally.

The **r** and **w** commands take an optional *rfile* (or *wfile*) parameter, separated from the command letter by one or more <blank>s; implementations may allow zero separation as an extension.

The argument *rfile* or the argument *wfile* terminates the command line. Each *wfile* is created before processing begins. Implementations support at least nine *wfile* arguments in the script; the actual number (≥9) that is supported by the implementation is unspecified. The use of the *wfile* parameter causes that file to be initially created, if it does not exist, or replaces the contents of an existing file.

The `b`, `r`, `s`, `t`, `w`, `y`, `!`, and **:** commands accept additional arguments. The following synopses indicates which arguments are separated from the command by a single <space>.

Two of the commands take a *command-list*, which is a list of `sed` commands separated by <newline>s, as follows:

```
{command
command
...
}
```

The `{` can be preceded with `<blank>`s and can be followed with white space. The commands can be preceded by white space. The terminating `}` is preceded by a `<newline>` and then zero or more `<blank>`s.

`[`*2addr*`] {`*command-list*`}`
: Execute *command-list* only when the pattern space is selected.

`[`*1addr*`]a` *text*
: Write *text* to standard output just before each attempt to fetch a line of input, whether by executing the `N` command or by beginning a new cycle.

`[`*2addr*`]b [`*label*`]`
: Branch to the : command bearing the *label*. If *label* is not specified, branch to the end of the script. The implementation shall support *labels* recognized as unique up to at least 8 characters; the actual length (8) that shall be supported by the implementation is unspecified. It is unspecified whether exceeding a label length causes an error or a silent truncation.

[*2addr*]`c` *text*
: Delete the pattern space. With `0` or `1` address or at the end of a 2-address range, place *text* on the output.

`[`*2addr*`]d`
: Delete the pattern space and start the next cycle.

`[`*2addr*`]D`
: Delete the initial segment of the pattern space through the first `<newline>` and start the next cycle.

`[`*2addr*`]g`
: Replace the contents of the pattern space by the contents of the hold space.

`[`*2addr*`]G`
: Append to the pattern space a `<newline>` followed by the contents of the hold space.

`[`*2addr*`]h`
: Replace the contents of the hold space with the contents of the pattern space.

`[`*2addr*`]H`
: Append to the hold space a `<newline>` followed by the contents of the pattern space.

`[`*2addr*`]i` *text*
: Write text to standard output.

`[`*2addr*`]l` (The letter ell.) Write the pattern space to standard output in a visually unambiguous form. The characters listed in the following table are written as the corresponding escape sequence. Non-printable characters not in this table are written as one three-digit octal number (with a preceding backslash) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than nine bits, the format used for non-printable characters is implementation defined.

Long lines are folded with the point of folding indicated by writing `<backslach><newline>`; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line is marked with a `$`.

Escape Sequence	Represents Character	Terminal Action
`\\`	`backslash`	None.
`\a`	`<alert>`	Attempts to alert the user through audible or visible notification.
`\b`	`<backspace>`	Moves the printing position to one column before the current position, unless the current position is the start of a line.
`\f`	`<form-feed>`	Moves the printing position to the initial printing position of the next logical page.
`\n`	`<newline>`	Moves the printing position to the start of the next line.
`\r`	`<carriage-return>`	Moves the printing position to the start of the current line.
`\t`	`<tab>`	Moves the printing position to the next tab position on the current line. If there are no more tab positions left on the line, the behavior is undefined.
`\v`	`<vertical tab>`	Moves the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behavior is undefined.

[*2addr*]n	Write the pattern space to standard output if the default output has not been suppressed, and replace the pattern space with the net line of input.
[*2addr*]N	Append the next line of input to the pattern space, using an embedded `<newline>` to separate the appended material from the original material. Note that the current line number changes.
[*2addr*]p	Write the pattern space to standard output.
[*2addr*]P	Write the pattern space, up to the first `<newline>`, to standard output.
[*2addr*]q	Branch to the end of the script and quit without starting a new cycle.
[*2addr*]r *rfile*	Copy the contents of *rfile* to standard output just before each attempt to fetch a line of input. If *rfile* does not exist or cannot be read, it shall be treated as if it were an empty file, causing no error condition.

[*2addr*]s /*regular expression*/*replacement*/*flags*

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character other than backslash or `<newline>` can be used instead of a slash to delimit the RE and the replacement. Within the RE and the replacement, the RE delimiter itself can be used as a literal character if it is preceded by a backslash.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE. The special meaning of an & in this context can be suppressed by preceding it by a backslash. The characters *n*, where *n* is a digit, are replaced by the text matched by the corresponding backreference expression. For each backslash encountered in scanning *replacement* from beginning to end, the following character loses its special meaning (if any). It is unspecified what special meaning is given to any character other than &, \, or digits.

A line can be split by substituting a `<newline>` character into it. The application escapes the `<newline>` in the replacement by preceding it by backslash. A substitution is considered to have been performed even if the replacement string is identical to the string that it replaces.

The value of *flags* is zero or more for one of the following:

`n` Substitute for the *n*th occurrence only of the *regular expression* found within the pattern space.

`g` Globally substitute for all non-overlapping instances of the *regular expression* rather than just the first one. If both `g` and `n` are specified, the results are unspecified.

`p` Write the pattern space to standard output if a replacement was made.

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

`[`*2addr*`]t [`*label*`]`
Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a `t`. If *label* is not specified, branch to the end of the script.

`[`*2addr*`]w` *wfile*
Append [write] the pattern space to *wfile*.

`[`*2addr*`]x` Exchange the contents of the pattern and hold spaces.

`[`*2addr*`]y /`*string1*`/`*string2*`/`
Replace all occurrences of characters in *string1* with the corresponding characters in *string2*. If the number of characters in *string1* and *string2* are not equal, or if any of the characters in *string1* appear more than once, the results are undefined. Any character other than backslash or `<newline>` can be used instead of slash to delimit the strings. Within *string1* and *string2*, the delimiter itself can be used as a literal character if it is preceded by a backslash.

[*2addr*]!*command*

[*2addr*]!{*command-list*}
: Apply the *command* or *command-list* only to the lines that are not selected by the address(es).

[*0addr*]:*label*
: This command does nothing; it bears a label for the `b` and `t` commands to branch to.

[*0addr*]=
: Write the following to standard output:

```
"%d\n", <current line number>
```

[*0addr*]
: An empty command is ignored.

[*0addr*]#
: The `#` and the remainder of the line are ignored (treated as a comment), with the single exception that if the first two characters in the file are `#`*n*, the default output is suppressed; this is the equivalent of specifying **-n** on the command line.

DIAGNOSTICS

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

`0`
: Successful completion.

`>0`
: An error occurred.

COPYRIGHT

NAME

`sems` – user semaphore status and removal

SYNOPSIS

`sems`

`sems -r` *name ...*

DESCRIPTION

`sems` shows the status of all user semaphores on the system. For each semaphore, `sems` displays the address, the current count, and the name.

NOTE

Count is negative if there are any waiters.

OPTIONS

`-r` Each semaphore named is removed, if possible.

SEE ALSO

System Calls - `sem_get()`, `sem_count()`

NAME

`setactive` – set the active (boot) partition

SYNOPSIS

`setactive` *device_name partition_name*

DESCRIPTION

`setactive` sets the active (boot) disk partition. The arguments passed to this command are the device name (without any partition) *device_name* and the partition to be chosen (i.e., `A` to `D`). For example, to choose partition A of device `/dev/hd0` to be the active partition, the command should be

```
setactive /dev/hd0 A
```

and not

```
setactive /dev/hd0A
```

DIAGNOSTICS

`setactive` fails for the following reasons:

- If the specified *device_name* does not exist.
- If the specified *device_name* is not a disk.
- Wrong usage or failure to specify required parameters.
- If the specified block/track to be added does not exist.
- If the specified file *pathname* cannot be opened for saving the bad block list.
- If all the disk blocks have been used by initial partitions and the command asks that another partition be added.

SEE ALSO

Utility Program - `mkpart`

NAME

`setprio` – alter priority of a process

SYNOPSIS

`setprio [ +- ]` *priority process ID ...*

DESCRIPTION

`setprio` sets the CPU priority of each process to priority. The new priority should be a value between 0 and 255, inclusive. Higher values indicate higher priority. If the priority value is preceded by a plus or minus sign, the current priorities of the processes are modified by the given amount.

Note that under LynxOS, a high priority process which is not waiting on a resource can only be preempted by a process with the same or higher priority. The operating system never adjusts priorities for the benefit of interactive processes.

SEE ALSO

Utility Programs - `dlsh, ps`

System Calls - `getpriority(), setpriority()`

NAME

`show` – show hex and character dump of files

SYNOPSIS

show ([-b *offset* **| +b] [-e** *offset* **| +e | -l** *lines* **| -c** *count***]**
[+a *offset* **| -a** *offset***]** *file ...* **] ...**

DESCRIPTION

`show` is binary dump utility that displays a combination hex dump / character dump in a compact form. Multiple file names can be given to `show`. If options are given between filenames, then such options will be applicable only to those files which follow them.

OPTIONS

-b *offset* Show from the given *offset* in the file.

+b Show from the beginning of the file. (Useful in resetting the effect of an earlier **-b** that was meant only for a previous file.)

-e *offset* Show up to the given *offset* in the file.

+e Show up to the end of file. (Useful in resetting the effect of an earlier **-e** that was meant only for a previous file.)

-l *lines* Show only the given number of (display) lines from the file. (Each display line contains up to 16 bytes from the file.)

-c *count* Show only the given number of characters from the file.

+a *offset* Normally the display offsets will be file offsets. With this option, the display offset is modified by adding the *offset* to the file offset.

-a *offset* Similar to **+a**, except that *offset* is subtracted from the file *offset*.

For the **-b**, **-e**, **+a** and **-a** options, *offset* is always interpreted as hexadecimal, so there is no need to specify `0x`. For the **-l** and **-c** options, lines and count should be given with C style prefixing: `0X` for hexadecimal, `0` for octal, and no prefix for decimal. Negative numbers given to the `-l` and `-c` options are ignored.

SEE ALSO

Utility Programs - `bined, binfix, flook, od`

NAME

showmount – show all remote mounts

SYNOPSIS

showmount [-me] [*host* **]**

DESCRIPTION

showmount lists all the clients that have remotely mounted a file system from *host*. This information is maintained by the mountd server on *host*, and is saved across crashes in the file /etc/rmtab. The default value for host is the value returned by hostname.

OPTIONS

–m Print all remote mounts in the format hostname directory; where hostname is the name of the client, and directory is the root of the file system that has been mounted.

–e Print the list of exported file systems.

FILES

/etc/rmtab

SEE ALSO

Utility Programs - hostname, mountd

File Formats - exports

NAME

`shownode` – show inode contents

SYNOPSIS

`shownode` *device inum* **`[-m`** *mode* **`]`** ...

`shownode` *file*

DESCRIPTION

`shownode` displays the contents of selected inodes from a file system on a device. If an inode number is followed by **`-m`** and an octal number, the mode field of the inode is set to the given value.

`shownode` will also display the inode for any file, given the file name.

NOTE

In certain cases the information reported by `shownode` is incorrect. Specifically, if a file is open and being modified, the information is stored in memory by the operating system and only written to disk (the source of information for `shownode`) when the file is closed or when `sync` is called.

SEE ALSO

Utility Programs - `ls, flook`

System Calls - `stat()`

NAME

size – report in bytes the sectional and total sizes for an object file.

SYNOPSIS

size [-A|-B| --format=*compatibility***] [--help]**
[-d|-o|-x|--radix=*number***] [--target=***bfdname***] [-V|--version]**
objfile...

DESCRIPTION

The GNU size utility lists the section sizes—and the total size—for each of the object files *objfile* in its argument list. By default, one line of output is generated for each object file or each module in an archive.

OPTIONS

-A
-B
--format *compatibility*
Using one of these options, you can choose whether the output from GNU size resembles output from System V size (using '**-A**', or '--format=sysv'), or Berkeley size (using '**-B**', or '--format=berkeley'). The default is the one-line format similar to Berkeley's.

--help Show a summary of acceptable arguments and options.

-d
-o
-x
--radix *number* Using one of these options, you can control whether the size of each section is given in decimal ('**-d**', or '--radix 10'); octal ('**-o**', or '--radix 8'); or hexadecimal ('**-x**', or '--radix 16'). In '**--radix** *number*', only the three values (8, 10, 16) are supported. The total size is always given in two radices; decimal and hexadecimal for '**-d**' or '**-x**' output, or octal and hexadecimal if you're using '**-o**'.

--target *bfdname*
: You can specify a particular object-code format for *objfile* as *bfdname*. This may not be necessary; `size` can automatically recognize many formats. See `objdump(1)` for information on listing available formats.

-V
--version
: Display version number information on `size` itself.

SEE ALSO

Utility Programs - `ar`, `objdump`

Documents - *The GNU Binary Utilities*, Roland H. Pesch.

COPYING

NAME

`slattach` – attach serial lines as network interfaces

SYNOPSIS

`slattach` *ttyname* **[** *baudrate* **]**

DESCRIPTION

`slattach` is used to assign a `tty` line to a network interface, and to define the network source and destination addresses. The *ttyname* parameter is a string of the form `ttyXX`, or `/dev/ttyXX`. The optional *baudrate* parameter is used to set the speed of the connection. If not specified, the default of 9600 is used.

Only the super-user may attach a network interface.

To detach the interface, use

`ifconfig` *interface-name* `down`

after killing off the `slattach` process. interface-name is the name that is shown by `netstat`.

EXAMPLES

```
slattach /dev/ttyd0
```

```
slattach /dev/ttyd0 19200
```

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

Utility Programs - `netstat, intro, ifconfig`

NAME

slaveboot – transfers a kernel image to a diskless SCMP client

SYNOPSIS

slaveboot [-i *ifname***] [-f** *cfgfile***] [-l** *loadaddr***] [-r** *runaddr***]** *kdifile host*

DESCRIPTION

slaveboot is part of the Snuggly Coupled Multiprocessing (SCMP) support of LynxOS. After creating a Kernel Downloadable Image (KDI) for a diskless SCMP client, slaveboot is needed to copy the KDI over the VMEbus into the local memory of a slave CPU board and to have the remote CPU board start execution.

The VMEbus slave address, the download address and execution offset are determined based upon the contents of the SCMP configuration file, which is usually /etc/bplane.conf. The slaveboot program will search the configuration file for the correct parameters for the supplied *host*. Based on the *type* parameter (see the descriptions of the **-l** and **-r** options and see bplane.conf(5)) for a given *host*, certain of the default values offered by slaveboot will differ.

OPTIONS

-f *cfgfile* Specifies an explicit database file (path *cfgfile*) from which to read configuration information for the SCMP system. The default is /etc/bplane.conf.

-i *ifname* Specifies an explicit interface name for the SCMP interface. The default is bplane0.

-l *loadaddr* Specifies a particular load address. For boards of type 1 (68k), the default load address is 0x10000. For boards of type 2 (PowerPC), the default load address is 0x4000.

-r *runaddr* Specifies a particular run address. For boards of type 1 (68k), the default run address is 0x10020. For boards of type 2 (PowerPC), the default run address is 0x4020.

kdifile	Specifies a Kernel Downloadable Image (KDI) as created by the `mkimage` utility.
host	Specifies which diskless SCMP participant should be booted. *host* is a symbolic name or IP address identifying one entry in the SCMP configuration file.

FILES

`/etc/bplane.conf`

`/etc/hosts`

SEE ALSO

Utility Programs – `mkimage`, `bpconfig`

File Formats – `bplane.conf`

NAME

`sleep` – suspend a current process for a given interval

SYNOPSIS

`sleep` *seconds*

DESCRIPTION

`sleep` does nothing at all for the given integral number of seconds. `sleep` is useful for shell scripts which require an event to occur after a certain time has elapsed. Because the process waits for a signal, it does not consume system resources while sleeping.

NAME

`smems` – shared memory status and removal

SYNOPSIS

smems

smems -r *name ...*

smems -d *name ...*

DESCRIPTION

`smems` shows the status of all the user-created shared memory segments on the system. For each segment, `smems` displays the number of users, the size, and the name of the segment.

OPTIONS

-r Removes each named shared memory segment, if possible.

-d The contents of each named segment are written to standard output.

SEE ALSO

Utility Programs - `ps`

System Calls - `smem_get(), smem_remove()`

NAME

`snmpd` – LynxSNMP master agent daemon

SYNOPSIS

`/net/snmpd` *config_file scratch_file*

DESCRIPTION

The `snmpd` daemon is the master Simple Network Management Protocol (SNMP) agent. It is invoked within `/etc/config/netservice.snmp` as:

```
/net/snmpd /etc/config.snmp /etc/NOV
```

The SNMP installation script configures the `/etc/sysconfig` file to automatically begin the `snmpd` daemon.

The file `/etc/config.snmp` contains the configuration file for the Master agent daemon and the file `/etc/NOV` serves as a temporary or scratch file. The `snmpd` daemon supports the following groups under the SNMP Management Information Base (MIB):

```
# The mandatory SNMP group under MIB-2
iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).snmp(11)

# Support for SNMP Multiplexing Protocol support (Agent/Subagent
support)
iso(1).org(3).dod(6).internet(1).private(4).enterprise(1)

# The system group under MIB-2
iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).system(1)
```

The `snmpd` daemon also supports basic SNMPv2 MIB and SNMPv2 Party MIB under the group:

```
iso(1).org(3).dod(6).internet(1).snmpv2(6).snmpModules(3)
```

Upon execution, `snmpd` reads the configuration file `/etc/config.snmp`. This file contains information about the Trap Manager, the Community string and Access permissions. The file `/etc/services` contains the default port numbers used by `snmpd`.

SEE ALSO

Utility Programs - `mibIIagent`, `lynxos_agent`

File Formats - `config.snmp`

NAME

`sort` – sort or merge files

SYNOPSIS

sort [-bcdfimnrTu -t*x* **-o** *file***] [** *keys***] [** *files***]**

DESCRIPTION

`sort` orders lines from each of the given input files (or standard input if none are given) and writes the resultant list to standard output. Unless instructed otherwise, `sort` compares entire lines and arranges them in ascending character set order. These global options alter the default behavior:

-b Ignore leading blanks and tabs.

-c Produce a message if the input files are not already sorted. No other action is taken.

-d Sort files in “dictionary” order. Only upper and lower case letters, digits, and blanks are considered.

-f Causes `sort` to treat upper and lower case letters as identical. Upper case letters are mapped into lower case equivalents, so punctuation characters whose values lie between the upper and lower case letters in the collating sequence will now appear to be smaller than the upper case letters.

-i Ignore non-ASCII characters (characters whose value lies above the last ASCII character).

-m Merge the already sorted input files.

-n A floating-point numeric value is parsed starting at the first non-blank character of each line, and used as the sort key. The number can be preceded by a minus sign.

-r Reverse the order of comparisons

`sort` can be instructed to use one or more fields from the line as keys, rather than the entire line. Arguments of the form + *pos1* and - *pos2* describe a key beginning at position *pos1* and ending at position *pos2*. A position is given as *field.byte*, representing a field number (the first field is field 0) and a byte offset into the field. Fields are by default separated by whitespace, though this can be changed with an option described below. Thus the pair `+2.3 -3` describes the key beginning three bytes into field 2 and ending just before field 3. A missing - *pos* indicates that the key continues to the end of the line.

The global option flags described earlier can be applied individually to keys by appending them to a + *pos* or - *pos* descriptor. Keys which have no explicit flags inherit the global flags.

The order in which keys are presented to `sort` dictates the order in which they are examined. If two lines are identical in terms of all keys given, they are ordered by comparing them in their entirety using any global flags.

OPTIONS

-o *output* Use the file specified by *output* for output, instead of standard output.

-t*x* Set the field separator to the character *x*. If *x* is omitted, tab is assumed.

-u Discard all but one line in a sequence of multiple, equal lines for comparison. All ignored bytes, and bytes outside the keys, do not participate in the comparison.

EXAMPLES

`sort -uf list` Sorts a list of lines, treating upper and lower case letters equally. The resulting list will contain no duplicates.

`sort -t: +2n /etc/passwd`
Sort all the lines in the password file based on the user ID numbers which begin in the third colon-separated field.

`sort -um +0 -1 dates`
Output the first occurrence of each month in a previously sorted file of month-day entries. The options `-um` specified with just one input file make the choice of a unique month from a set of equal lines predictable.

FILES

`/usr/tmp/stm*`	First try for temporary file.
`/tmp/*`	Second try for temporary files.

NOTE

Lines longer than 512 bytes are truncated silently.

SEE ALSO

Utility Programs - `uniq`

NAME

`sparcboot` – primary kernel boot program (SPARC)

SYNOPSIS

`sparcboot`

`makeboot`

`openboot`

DESCRIPTION

`sparcboot` is the primary boot loader for LynxOS systems running on machines that support the Open Boot PROM. `sparcboot` is installed on a bootable disk using the `makeboot` command and is invoked as part of the booting sequence. A disk *must* be partitioned and labeled with a SunOS-compatible label in order to be booted from the Open Boot PROM (in contrast, LynxOS systems that do not use the Open Boot PROM can generally be booted from un-partitioned disks). See `mkpart` (SPARC version) for more details.

The booting process loads in `sparcboot`, which loads in `openboot`, the secondary loader. `openboot` then loads LynxOS and jumps to its start address.

SEE ALSO

Utility Program - `makeboot` (SPARC version), `openboot`, `mkpart` (SPARC version)

NAME

split – split a file into pieces

SYNOPSIS

split [–*number* **] [** *file* **[** *name* **]]**

DESCRIPTION

split reads *file* and writes it in *number*-line pieces (default 1000 lines) onto a set of output files called *name***aa**, *name***ab**, ..., *name***zz** (a maximum of 676 files). If no output *name* is given, **x** is used as the default (output files will be called **xaa**, **xab**, etc.). If no input *file* is given, or if **–** is given as the input *file*, then the standard input file is used.

OPTIONS

–*number* The number of lines in each piece.

SEE ALSO

Utility Programs - cat

NAME

`ssplit` – break stream into pieces

SYNOPSIS

ssplit [-cf] [-v *volsize* **-b** *blocksize* **-B** *bufsize* **]** *file*

DESCRIPTION

`ssplit` reads standard input and produces a set of output files, each of a specified size. The output files are given names constructed by adding a dot and a sequence number to *file*. Alternatively, `ssplit` can be used to copy a file onto several volumes on a removable-media device, such as a floppy disk drive.

OPTIONS

-c Instructs `ssplit` to prompt for a media change and wait for operator response. If this option is chosen, the output file name is not changed between volumes.

-f Forces output blocks to be a multiple of *blocksize* in size. Necessary when the output file is a raw interface to a block device.

-v *volsize* Sets the output volume size to *volsize*. The *volsize* value may include a trailing size specifier as follows:

M or **m**	Size is in megabytes.
K or **k**	Size is in kilobytes.
B or **b**	Size is in blocksize-byte blocks.

If no size specifier is given, the size is assumed to be in bytes. If no **-v** is given at all, `ssplit` will write until an `ENOSPC` error is generated; that is, until the device is full. In this case, **-c** is assumed.

-b *blocksize* Specifies blocksize to be used when **-f** is in effect, or when calculating volume size. If not given, *blocksize* is 512 bytes.

`-B` *bufsize* Specifies the size of output blocks in terms of *blocksize*-byte logical blocks. Default is 1 logical block.

NOTE

When writing to a floppy disk drive or a tape drive, it is advisable to use the raw interface instead of the block interface. For example, to copy `bigfile` to some number of floppy disks:

```
ssplit < bigfile /dev/rfd0
```

SEE ALSO

Utility Programs - `tar, cp`

NAME

`st` – post-mortem debugging aid

SYNOPSIS

st [-c *corefile***] [-a] [** *a.outfile***]**

DESCRIPTION

`st` combines information from a core dump file (either *corefile* or `core` by default) and the executable file whose misfortune it was to produce it. It then prints a trace of functions active at the time of the event which generated the core dump, including the current program counter value. It also gives the limits and current extent for the stack and data areas. Finally, `st` prints the contents of all the registers at the time of the core dump. The contents of those registers which may contain addresses are matched against the symbol table from the executable file and printed symbolically if possible.

With no arguments, `st` attempts to open `core` and then to open the executable file named therein. A specific executable file *a.outfile* can be given, usually only necessary when the small 32-byte pathname limitation imposed by the operating system on the core file format cannot hold the entire actual pathname. A different core file can be given with the **-c** switch, necessary if the dump was left in a directory other than the current one.

If `st` is unable to read the symbol table from the executable, or if it cannot find the executable file, it will print just the addresses without the associated function names. The **-a** option can be used to force this kind of output, if desired.

`st` is capable of examining core dumps from either `coff` or `a.out` executables.

SEE ALSO

Utility Programs - `disasm, dump`

File Formats - `a.out, core`

NAME

`strace` – print trace messages from STREAMS subsystem

SYNOPSIS

`strace [` *module-id sub-id tracing-priority* **`]`** ...

DESCRIPTION

`strace`, a daemon process which can be started at any time, is used to print out event trace messages coming from STREAMS drivers and modules. These event traces come from the STREAMS log driver, `strlog`.

Each message output is of the form:

sequence time ticks priority flags module-id sub-id msg

where each field is defined as follows:

sequence	sequence number of this trace
time	time of the event, formatted as *hours*`:`*minutes*`:`*seconds*
ticks	time of the event, in clock ticks since the system was booted
priority	event trace priority level
flags	various facts about this trace, indicated by letters:
	`T` Event trace message
	`E` This message is also in the error log
	`F` Fatal error
	`N` System administrator notified
module-id	ID of source module
sub-id	sub-ID of source
msg	text of the event trace message

If no arguments are given, `strace` prints all such event trace messages from all STREAMS modules and drivers.

Optionally, you can invoke `strace` with ordered triples of arguments, *module-id sub-id priority*. Each set of three arguments indicates a particular component of the STREAMS subsystem to retrieve the event traces from. The *module-id* indicates a particular module/driver, and the *sub-id* indicates a sub-ID within same. Event traces of priority less than or equal to the *priority* will be retrieved. For any of *module-id*, *sub-id*, or *priotiry*, you can substitute `all` if you wish no restriction based on that particular component.

`strace` is a daemon process. Once started, it runs continuously in the background of the system.

EXAMPLES

To print out all trace messages from module 86, trace level less than or equal to 13:

```
strace 86 all 13
```

NAME

`strclean` – clean error messages from STREAMS subsystem

SYNOPSIS

strclean [-d *logdir* **] [-a** *age* **]**

DESCRIPTION

Use `strclean` to clean up the error logging directory used by STREAMS to deposit its error messages.

If invoked with no arguments, `strclean` deletes all files named `error.*` in the default STREAMS error logging directory (`/usr/adm/streams`) which have not been modified in the last 3 days.

OPTIONS

-d *directory*	You can change the directory cleaned by specifying a different directory with the **-d** option.
-a *age*	You can change the time threshold for removing error files from 3 days to some other number of days by specifying a different number of days with the **-a** option.

EXAMPLES

To clean out all error logs which are older than 13 days, type:

```
strclean -a 13
```

To clean out all error logs older than 2 days in the directory `/usr/adm/streams.old`, type:

```
strclean -d /usr/adm/streams.old -a 2
```

FILES

`/usr/adm/streams/error.*`

SEE ALSO

Utility Programs - `strerr`

NAME

`strerr` – STREAMS error logger

SYNOPSIS

`strerr`

DESCRIPTION

`strerr`, a daemon process which can be started at any time, logs error messages from the STREAMS log driver, `strlog`, and adds them to a file. The files produced by `strerr` live in the directory `/usr/adm/streams/error.`*month-day*, where *month* and *day* are numbers indicating the day on which the errors occurred.

Each error message is of the form:

sequence time ticks priority flags module-id sub-id msg

where each field is defined as follows:

sequence	sequence number of this error
time	time of the error, formatted as *hours*`:`*minutes*`:`*seconds*
ticks	time of the error, in clock ticks since the system was booted
flags	various facts about this message, indicated by letters:
	`T` Event trace message
	`E` Error log message
	`F` Fatal error
	`N` Notify system administrator
module-id	ID of source module
sub-id	sub-ID of source
msg	text of the error message

These error messages generally reflect exceptions that necessitate the system administrator's attention. The **F** flag indicates total and utter failure of the associated STREAMS module. The **N** flag indicates that the sysadmin should be notified of this error.

`strerr` is a daemon process. Once started, it runs continuously in the background of the system.

FILES

`/usr/adm/streams/error.`*month-day*

SEE ALSO

Utility Programs – `strclean`

Driver Service Calls - `strlog`

NAME

`strings` – print the strings of printable characters in files

SYNOPSIS

strings [-a|-|--all] [-f|--print-file-name] [-o] [--help] [-v|--version] [-n *min-len***|-***min-len***|--bytes=***min-len***] [-t {o,x,d}[--target=***bfdname***] |--radix={o,x,d}]** *file*

DESCRIPTION

For each *file* given, GNU `strings` prints the printable character sequences that are at least 4 characters long (or the number given with the options below) and are followed by an unprintable character. By default, it only prints the strings from the initialized and loaded sections of object files; for other types of files, it prints the strings from the whole file.

`strings` is mainly useful for determining the contents of non-text files.

OPTIONS

The long and short forms of options, shown here as alternatives, are equivalent.

-a
--all
- Do not scan only the initialized and loaded sections of object files; scan the whole files.

-f
--print-file-name
Print the name of the file before each string.

--help Print a summary of the options to `strings` on the standard output and exit.

-v
--version Print the version number of `strings` on the standard output and exit.

-n *min-len*
-*min-len*
-bytes=*min-len* Print sequences of characters that are at least *min-len* characters long, instead of the default 4.

-t {o,x,d}
--radix={o,x,d} Print the offset within the file before each string. The single character argument specifies the radix of the offset—octal, hexadecimal, or decimal.

--target=*bfdname*
Specify an object code format other than your system's default format. See `objdump(1)`, for information on listing available formats.

-o Like `-to`.

SEE ALSO

Utility Programs - `ar, nm, objdump, ranlib`

Documents - '`binutils`' entry in `info`;

The GNU Binary Utilities, Roland H. Pesch

COPYING

NAME

`strip` – Discard symbols from object files.

SYNOPSIS

strip [-F *bfdname* **| --target=***bfdname***]**
[-I *bfdname* **| --input-target=***bfdname***]**
[-O *bfdname* **| --output-target=***bfdname***]**
[-R *sectionname* **| --remove-section=***sectionname***]**
[-s | --strip-all] [-S | -g | --strip-debug]
[--strip-unneeded]
[-x | --discard-all] [-X | --discard-locals]
[-K *symbolname* **| --keep-symbol=***symbolname***]**
[-N *symbolname* **| --strip-symbol=***symbolname***]**
[-v | --verbose]
[-V | --version] [-V | --help] *objfile*...

DESCRIPTION

GNU `strip` discards all symbols from the object files *objfile*. The list of object files may include archives. At least one object file must be given.

`strip` modifies the files named in its argument, rather than writing modified copies under different names.

OPTIONS

-F *bfdname*
--target=*bfdname*
: Treat the original *objfile* as a file with the object code format *bfdname*, and rewrite it in the same format.

--help
: Show a summary of the options to `strip` and exit.

-I *bfdname*
--input-target=*bfdname*
: Treat the original *objfile* as a file with the object code format *bfdname*.

-O *bfdname*
--output-target=*bfdname*
: Replace *objfile* with a file in the output format *bfdname*.

-R *sectionname*
--remove-section=*sectionname*
Remove the named section from the file. This option may be given more than once. Note that using this option inappropriately may make the object file unusable.

-s
--strip-all Remove all symbols.

-S
-g
--strip-debug Remove debugging symbols only.

--strip-unneeded
Strip all symbols that are not needed for relocation processing.

-N *symbolname*
--strip-symbol=*symbolname*
Remove symbol *symbolname* from the source file. This option may be given more than once, and may be combined with other strip options.

-x
--discard-all Remove non-global symbols.

-X
--discard-locals
Remove compiler-generated local symbols. (These usually start with "`L`" or "`.`".)

-K *symbolname*, **--keep-symbol=***symbolname*
Copy only symbol *symbolname* from the source file. This option may be given more than once.

-N *symbolname*, **--strip-symbol=***symbolname*
Do not copy symbol *symbolname* from the source file. This option may be given more than once, and may be combined with strip options other than **-K**.

-v
--verbose Verbose output: list all object files modified. In the case of archives, `strip -V` lists all members of the archive.

-V
--version Show the version number for `strip` and exit.

SEE ALSO

Documents - *The GNU Binary Utilities*, Roland H. Pesch.

COPYING

NAME

`stty` – set RS232 driver options and parameters

SYNOPSIS

`stty` *commands*

DESCRIPTION

`stty` sets various options for `tty` devices. If given the option `all`, `stty` reports all current configuration information.

COMMANDS

`all`	Report status of all normally used option settings. This option can appear more than once in the list of options, and reports current settings up to the point of its occurrence.
`even`	Set even parity checking on input.
`-even`	Turn off even parity checking.
`odd`	Set odd parity checking on input.
`-odd`	Turn off odd parity checking.
`raw`	Set raw mode. No input or output processing is performed.
`-raw`	Same as `cooked`.
`cooked`	Enable all normal input/output processing, including input line editing.
`cbreak`	Disable all input/output processing except signal generation and start/stop control.
`-nl`	The carriage return character is translated into the newline character on input. Also, the newline character is translated to a carriage return followed by a newline on output.

`nl`	Newline and carriage returned are accepted and output without translation.
`echo`	Echo every character typed.
`-echo`	Do not echo typed characters.
`lcase`	Map upper case characters to lower case characters.
`-lcase`	Do not map upper case characters to lower case characters.
`tandem`	Enable input flow control. This option allows the system to send a stop character out to a peripheral when the internal input queue is in danger of overflowing. The system will automatically send out a start character when it detects that its input queue has emptied to a sufficient extent. Most useful when the `tty` is connected to an active host, such as another computer.
`-tandem`	Do not provide input flow control.
`-tabs`	Replace tabs with spaces on output.
`tabs`	Leave tabs alone (output them as normal characters).

In the following option descriptions, the c argument can be a single character or it can be two character sequence (such as `^X`) for specifying control characters (delete, ASCII 127, is given as `^?`). In addition, if **c** is given as **u** or `undef`, it is made undefined.

`erase` *c*	Set the erase character to *c*. Usually set to `^H`.
`kill` *c*	Set the line kill character to *c*. Usually set to **`^U`**.
`intr` *c*	Set the interrupt character to *c*. Usually set to `^C`.
`quit` *c*	Set the quit character to *c*. Usually set to `^\`.
`start` *c*	Set the start character to *c*. Usually set to `^Q`.
`stop` *c*	Set the stop character to *c*. Usually set to `^S`.
`eof` *c*	Set the end-of-file character to *c*. Usually set to `^D`.
`brk` *c*	Set the break character to *c*. The brk character functions as an alternate line terminator.

`susp` *c*	Set the suspend-process character to *c*. Usually set to `^Z`.
`dsusp` *c*	Set delayed suspend-process character to *c*. Usually set to `^Y`.
`rprnt` *c*	Set line reprint character to *c*. Usually set to `^R`.
`flush` *c*	Set the flush output character to *c*. Usually set to `^O`.
`werase` *c*	Set the word erase character to *c*. Usually set to `^W`.
`lnext` *c*	Set the literal next character to *c*. Usually set to `^V`.

The following options set terminal characteristics which are usually of a more permanent nature. These are generally set by `tset` or by `stty` in a shell initialization file. They may of course be changed at any time as needed.

`75,110,134.5,150,300,600,1200,1800,2000,2400,4800,9600,19200`

Set terminal baud rate to given value.

`crt`	Set up typical options for a CRT (`crtbs`, `ctlecho`, `crterase`, and `crtkill`).
`crtbs`	Echo backspace whenever an erase character is typed.
`prterase`	Echo characters erased by the erase character within `\` and `/`. This option is useful on printing terminals.
`crterase`	Echo backspace-space-backspace whenever the erase character is typed.
`-crterase`	Do not echo erase sequence whenever erase character is typed, just backspace (character is still logically erased from the input queue).
`crtkill`	Wipe out (visually) entire input line on receipt of line kill character. Input is wiped out in accordance with the state of `crterase`.
`-crtkill`	Only echo line kill and newline on receipt of a line kill character.

`ctlecho`	Echo control characters as `^x` (delete is echoed as `^?`), where **x** is the character formed by adding 0100 to the control character's value.
`-ctlecho`	Echo control characters as-is.
`decctlq`	Only allow the start character to restart output after it has been stopped by the stop character.
`-decctlq`	Allow any character to start output after it has been stopped by the stop character.
`tostop`	Stop background jobs if they attempt to perform any output to the terminal.
`-tostop`	Allow background jobs to perform output to the terminal.
`pendin`	Do not flush input queue when switching between `cbreak` mode to cooked mode.
`-pendin`	Flush input queue upon performing a mode transition (i.e when changing from `cbreak` mode to `cooked` mode).
`mdmbuf`	Toggle between start and stop on carrier transitions.
`-mdmbuf`	Return error if a write is attempted after the carrier drops.
`litout`	Set literal output mode. No output translations are performed.
`-litout`	Do normal output processing.
`nohang`	Do not send the hangup signal if the carrier drops.
`-nohang`	Send the hangup signal to the control process group when the carrier drops.
`type` *config*	Set up the terminal according to configuration type config. The configuration information is recorded in the file `/etc/tconfig`

FILES

`/etc/tconfig`	Terminal configuration information file.

SEE ALSO

Utility Programs - `tty, tset`

System Calls - `ioctl()`

Library Functions - `ttyconfig()`

NAME

`su` – set effective user ID

SYNOPSIS

su [*username* **]**

DESCRIPTION

`su` requests a password for the account corresponding to *username* or `root` if *username* is omitted. If the password is correct, `su` sets the real and effective user ID and starts a subshell corresponding to the shell specified in the password file for *username* without changing directories. The new shell will not read *username*'s login initialization files because it is not a login shell. The user environment is left unchanged except for `HOME` and `SHELL`.

SEE ALSO

Utility Programs - `login`

System Calls - `setreuid(), setregid()`

NAME

`sync, syncer` – write out disk cache

SYNOPSIS

`sync`

`syncer [`*interval***`]`**

DESCRIPTION

`sync` updates the disk by arranging for data in memory-resident cache to be written. Note that `sync` simply initiates the cache cleanup; it is not guaranteed that the cache is completely flushed by the time `sync` finishes.

`syncer` is similar to `sync`, but it continues to update the cache periodically. It is usually placed in the background by a system initialization script. If a numeric *interval* is given, `syncer` will sleep that many seconds between flushes; the default interval is 30 seconds.

Note that when non-volatile caching is enabled, the `sync` programs usually need not be run.

SEE ALSO

System Calls - `sync()`

NAME

`syslogd, logit` – log systems messages

SYNOPSIS

syslogd [-f *config_file***] [-m** *mark_interval***] [-p** *log_socket***]**

logit [-n *logname***] [-p** *priority***] [-f** *inputfile***]**

DESCRIPTION

`syslogd` reads and logs messages to the system console, log files, other machines, and/or users as specified by its configuration file.

`syslogd` reads its configuration file when it starts up and whenever it receives a hangup signal. For information on the format of the configuration file, see `syslog.conf(5)`.

`syslogd` reads messages from the UNIX domain socket `/dev/log`, from an Internet domain socket specified in `/etc/services`, and from the special device `/dev/klog` (to read kernel messages).

`syslogd` creates the file `/etc/syslog.pid`, and stores its process id there. This can be used to kill or reconfigure `syslogd`.

The message sent to `syslogd` should consist of a single line. The message can contain a priority code, which should be a preceding decimal number in angle braces, for example, ‘<5.>’ This priority code should map into the priorities defined in the include file `<sys/syslog.h>`.

`logit` allows you to make system log entries from a shell script.

OPTIONS

The following are the options for `syslogd`:

-f *config_file*Specify the pathname of an alternate configuration file; the default is `/etc/syslog.conf`.

-m *mark_interval* Select the number of minutes between “mark” messages; the default is 20 minutes.

-p *log_socket*	Specify the pathname of an alternate log socket; the default is `/dev/log`.

The following are the options for `logit`:

-n *logname*	Use *logname* as the name of the log to open.
-p *priority*	Pass *priority* to `syslog` instead of `LOG_NOTICE`.
-f *inputfile*	Log data from *inputfile* instead of from the command line.

FILES

`/etc/syslog.conf`	The configuration file.
`/etc/syslog.pid`	The process id of current `syslogd`.
`/dev/log`	Name of the UNIX domain datagram log socket.
`/dev/klog`	The kernel log device.

SEE ALSO

Library Functions - `syslog()`

File Formats - `services()`, `syslog.conf()`

HISTORY

The `syslogd` command appeared in 4.3BSD.

COPYRIGHT

NAME

`tail` – print last few lines of a file

SYNOPSIS

tail [*options***] [** *file***]**

DESCRIPTION

`tail` copies to standard output lines from the end of the named *file*, or from standard input if no *file* is given. By default, the last ten lines are printed.

OPTIONS

–f `tail` prints the tail of the file, then waits for the file to grow, printing each new line as it appears. Useful for watching log files.

+ *n***[blc]** Begin output starting *n* blocks (**b**), lines (**l**), or characters (**c**) from the beginning of the file. Default unit is lines.

– *n***[blc]** Begin output starting *n* blocks, lines, or characters from the end of the file. Default unit is lines.

–r Print lines in reverse order. If no distance is specified, `tail` prints the entire file in reverse.

DIAGNOSTICS

Strange things may happen if `tail` is used with character special files.

SEE ALSO

Utility Programs - `less, head`

NAME

`talk` – converse interactively with another user

SYNOPSIS

`talk` *person* **[***ttyname***]**

DESCRIPTION

`talk` is a visual communication program which copies lines from your terminal to that of another user. Each character typed is transmitted immediately to the remote user. The screens of both parties are split in two windows: the top pane contains the characters you type, and the bottom contains those typed by the other user.

If you wish to talk to someone on your own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* takes one of the following forms:

```
host!user
host.user
host:user
user@host
```

though `user@host` is definitely preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, `talk` sends the message

`Message from TalkDaemon@`*other_machine...*
`talk: connection requested by` *your_name*`/@`*your_machine.*
`talk: respond with: talk` *your_name*`/@`*your_machine*

to the user you wish to talk to. At this point, the recipient of the message should reply by typing

`talk` *your_name*`/@`*your_machine*

It doesn't matter from which machine the recipient replies, as long as his login-name is the same. Once communication is established, the two parties may type simultaneously. Typing `CONTROL-L` (`^L`) will cause the screen to be reprinted, while the `erase`, `kill`, and `wordkill` characters will behave normally. To exit, type the `interrupt` character (usually `CONTROL-C`); `talk` then moves the cursor to the bottom of the screen and restores the terminal.

FILES

`/etc/hosts`	To find the recipient's machine.
`/etc/utmp`	To find the recipient's `tty`.
`/etc/inetd.conf`	Current `inetd` configuration file.

SEE ALSO

Utility Programs - `who, rwho`

NAME

`talkd` – remote user communication server

SYNOPSIS

`talkd`

DESCRIPTION

`talkd` is the server that notifies a user that somebody else wants to initiate a conversation. It acts as a repository of invitations, responding to requests by clients wishing to rendezvous by sending a `CTL_MSG` to the server of type `LOOK_UP` (see `<protocols/talkd.h>`). This causes the server to search its invitation tables to check if an invitation currently exists for the caller to speak to the respondent specified in the message.

If the lookup fails, the caller then sends an `ANNOUNCE` message causing the server to broadcast an announcement on the respondent's login ports requesting contact. When the respondent responds, the local server uses the recorded invitation to respond with the appropriate rendezvous address and the caller and respondent client programs establish a `STREAM` connection through which the conversation takes place.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `talk, inetd`

File Formats - `inetd.conf`

NAME

`tar -` GNU version of the tar archiving utility

SYNOPSIS

tar [-]A --catenate --concatenate | c --create | d --diff --compare | r --append | t --list | u --update | x -extract --get
[--atime-preserve]
[-b, --block-size *N* **]**
[-B, --read-full-blocks]
[-C, --directory *dir* **]**
[--checkpoint]
[--file [*hostname***:]***F* **]**
[--force- local]
[-F, --info-script *F* **--new-volume-script** *F* **]**
[-G, --incremental]
[-g, --listed-incremental *F* **]**
[-h, --dereference]
[-i, --ignore-zeros]
[--ignore-failed-read]
[-k, --keep-old-files]
[-K, --starting-file *F* **]**
[-l, --one-file-system]
[-L, --tape-length *N* **]**
[-m, --modification-time]
[-M, --multi-volume]
[-N, --after-date *date***, --newer** *date* **]**
[-o, --old-archive, --portability]
[-O, --to-stdout]
[-p, --same-permissions, --preserve-permissions]
[-P, --absolute-paths]
[--preserve]
[-R, --record-number]
[--remove-files]
[-s, --same-order, --preserve-order]
[--same-owner]
[-S, --sparse]
[-T, --files-from *F* **] [--null]**
[--totals]
[-v, --verbose]

[-V, --label *name* **]**
[-w, --interactive, --confirmation]
[-W, --verify]
[--exclude *file* **]**
[-X, --exclude-from *file* **]**
[-Z, --compress, --uncompress]
[-z, --gzip, --ungzip]
[--use-compress-program *prog* **]**
[--block-compress]
[-[0-7][lmh]] *filename1* **[***filename2,... filenameN***]**
directory1 **[***directory2,... directoryN***]**

tar --version

DESCRIPTION

This manual page documents the GNU version of `tar`, an archiving program designed to store and extract files from an archive file known as a tarfile. A tarfile may be made on a tape drive; however, it is also common to write a tarfile to a normal file. The first argument to tar must be one of the options: **Acdrtux**, followed by any optional functions. The final arguments to tar are the names of the files or directories which should be archived. The use of a directory name always implies that the subdirectories below should be included in the archive.

FUNCTION LETTERS

One of the following options must be used:

-A, **--catenate**, **--concatenate**
append tar files to an archive

-c, **--create** create a new archive

-d, --diff, --compare
find differences between archive and file system

--delete delete from the archive (not for use on mag tapes!)

-r, **--append** append files to the end of an archive

-t, **--list** list the contents of an archive

-u, **--update** only append files that are newer than copy in archive

-x, **--extract**, **--get**
extract files from an archive

OTHER OPTIONS

--atime-preserve
don't change access times on dumped files

-b, **--block-size***N*
block size of *N*x512 bytes (default *N*=20)

-B, **--read-full-blocks**
reblock as we read (for reading 4.2BSD pipes)

-C, **--directory** *dir*
change to directory *dir*

--checkpoint print directory names while reading the archive

-f, **--file** [*hostname*:]*F*
use archive file or device *F* (default `/dev/rmt0`)

--force-local archive file is local even if has a colon

-F, **--info-script** *F* **--new-volume-script** *F*
run script at end of each tape (implies **-M)**

-G, **--incremental**
create/list/extract old GNU-format incremental backup

-g, **--listed-incremental** *F*
create/list/extract new GNU-format incremental backup

-h, **--dereference**
don't dump symlinks; dump the files they point to

-i, **--ignore-zeros**
ignore blocks of zeros in archive (normally mean EOF)

--ignore-failed-read
don't exit with non-zero status on unreadable files

-k, **--keep-old-files**
keep existing files; don't overwrite them from archive

-K, **--starting-file** *F*
begin at file *F* in the archive

-l, **--one-file-system**
stay in local file system when creating an archive

-L, **--tape-length** *N*
change tapes after writing *N**1024 bytes

-m, **--modification-time**
don't extract file modified time

-M, **--multi-volume**
create/list/extract multi-volume archive

-N, **--after-date** *date*, **--newer** *date*
only store files newer than *date*

-o, **--old-archive**, **--portability**
write a V7 format archive, rather than ANSI format

-O, **--to-stdout**
extract files to standard output

-p, **--same-permissions**, **--preserve-permissions**
extract all protection information

-P, **--absolute-paths**
don't strip leading '/' characters from file names

--preserve like **-p -s**

-R, **--record-number**
show record number within archive with each message

--remove-files
remove files after adding them to the archive

-s, **--same-order**, **--preserve-order**
list of names to extract is sorted to match archive

--same-owner create extracted files with the same ownership

-S, **--sparse** handle sparse files efficiently

-T, **--files-from** *F*
get names to extract or create from file *F*

--null cause the **-T** option to accept null-terminated names; also disables the **-C** option.

--totals print total bytes written with **--create**

-v, **--verbose** verbosely list files processed

-V, **--label** *name*
create archive with volume name *name*

--version print tar program version number

-w, **--interactive**, **--confirmation**
ask for confirmation for every action

-W, **--verify** attempt to verify the archive after writing it

--exclude *file*
exclude file *file*

-X, **--exclude-from** *file*
exclude files listed in *file*

-Z, **--compress**, **--uncompress**
filter the archive through compress

-z, **--gzip**, **--ungzip**
filter the archive through `gzip`

--use-compress-program *prog*
filter the archive through *prog* (which must accept **-d**)

--block-compress
block the output of compression program for tapes

-[0-7][lmh] specify drive and density

EXAMPLES

To collect all files in a subdirectory called `dir` into an archive called `archive`:

```
tar -cf archive dir
```

To print the contents of the `archive` to standard output:

```
tar -tvf archive
```

To extract the files from the archive and create a duplicate of `dir` in the subdirectory `backup`:

```
cd backup
tar -xf ../archive
```

To compress the archive with `gzip` while creating it:

```
tar -czf archive dir
```

To extract archives that have been compressed with either `gzip` or `compress`:

```
tar -xzf archive.tar.Z
```

To copy the files files from the local machine to the directory on a remote machine:

tar -cf - *files* **| rsh** *machine* **"(cd** *dir***; tar xf -)"**

To extract all files on a tape device `/dev/rtape`:

```
tar -xvf /dev/rtape
```

NOTE

Previous releases of LynxOS referred to the GNU `tar` utility as "`gnutar`."

You can use GNU `tar` on archives created with the legacy Lynx `tar` (`l-tar`). However, `l-tar` should not be used on archives created with GNU `tar`.

NAME

`tcpdump` – dump traffic on a network

SYNOPSIS

tcpdump [-deflnNOpqStvx] [-c *count* **] [-F** *file* **]**
[-i *interface* **] [-r** *file* **] [-s** *snaplen* **]**
[-T *type* **] [-w** *file* **] [** *expression* **]**

DESCRIPTION

`tcpdump` prints out the headers of packets on a network interface that match the boolean *expression*. You must have read access to `/dev/bpf*`.

OPTIONS

-c *count* Exit after receiving *count* packets.

-d Dump the compiled packet-matching code in a human readable form to standard output and stop.

-dd Dump packet-matching code as a `C` program fragment.

-ddd Dump packet-matching code as decimal numbers (preceded with a count).

-e Print the link-level header on each dump line.

-f Print ‘foreign’ internet addresses numerically rather than symbolically (this option is intended to get around serious brain damage in Sun’s yp server — usually it hangs forever translating non-local internet numbers).

-F *file* Use *file* as input for the filter expression. An additional expression given on the command line is ignored.

-i *interface* Listen on *interface*. If unspecified, `tcpdump` searches the system interface list for the lowest numbered, configured up interface (excluding loopback). Ties are broken by choosing the earliest match.

-l Make stdout line buffered. Useful if you want to see the data while capturing it. For example,

```
tcpdump -l|tee dat
```

or

```
tcpdump -l > dat & tail -f dat
```

-n Don't convert addresses (i.e., host addresses, port numbers, etc.) to names.

-N Don't print domain name qualification of host names. E.g., if you give this flag then `tcpdump` will print "`nic`" instead of "`nic.ddn.mil`".

-O Do not run the packet-matching code optimizer. This is useful only if you suspect a bug in the optimizer.

-p *Don't* put the interface into promiscuous mode. Note that the interface might be in promiscuous mode for some other reason; hence, **-p** cannot be used as an abbreviation for

```
ether host {local-hw-addr} or ether
broadcast
```

-q Quick (quiet?) output. Print less protocol information so output lines are shorter.

-r *file* Read packets from *file* (which was created with the **-w** option). Standard input is used if *file* is "-".

-s *snaplen* Snarf *snaplen* bytes of data from each packet rather than the default of 68 (with SunOS's NIT, the minimum is actually 96). 68 bytes is adequate for IP, ICMP, TCP and UDP but may truncate protocol information from name server and NFS packets (see below). Packets truncated because of a limited snapshot are indicated in the output with "[|*proto*]", where *proto* is the name of the protocol level at which the truncation has occurred. Note that taking larger snapshots both increases the amount of time it takes to process packets and, effectively, decreases the amount of packet buffering. This may cause packets to be lost. You should limit *snaplen* to the smallest number that will capture the protocol information you're interested in.

-T *type* — Force packets selected by *expression* to be interpreted the specified *type*. Currently known types are `rpc` (Remote Procedure Call), `rtp` (Real-Time Applications protocol), `rtcp` (Real-Time Applications control protocol), `vat` (Visual Audio Tool), and `wb` (distributed White Board).

-S — Print absolute, rather than relative, TCP sequence numbers.

-t — *Don't* print a timestamp on each dump line.

-tt — Print an unformatted timestamp on each dump line.

-v — (Slightly more) verbose output. For example, the time to live and type of service information in an IP packet is printed.

-vv — Even more verbose output. For example, additional fields are printed from NFS reply packets.

-w *file* — Write the raw packets to *file* rather than parsing and printing them out. They can later be printed with the **-r** option. Standard output is used if *file* is "-".

-x — Print each packet (minus its link level header) in hex. The smaller of the entire packet or *snaplen* bytes will be printed.

expression — selects which packets will be dumped. If no *expression* is given, all packets on the net will be dumped. Otherwise, only packets for which *expression* is 'true' will be dumped.

EXPRESSIONS

The *expression* consists of one or more *primitives.* Primitives usually consist of an *id* (name or number) preceded by one or more qualifiers. There are three different kinds of qualifier:

type — qualifiers say what kind of thing the id name or number refers to. Possible types are `host`, `net` and `port`. E.g., '`host foo`', '`net 128.3`', '`port 20`'. If there is no type qualifier, `host` is assumed.

dir qualifiers specify a particular transfer direction to and/or from *id.* Possible directions are `src`, `dst`, `src or dst` and `src and dst`. For example,

```
src foo
dst net 128.3
src or dst port ftp-data
```

If there is no *dir* qualifier, `src or dst` is assumed. For 'null' link layers (i.e. point to point protocols such as slip) the `inbound` and `outbound` qualifiers can be used to specify a desired direction.

proto qualifiers restrict the match to a particular protocol. Possible protos are: `ether`, `fddi`, `ip`, `arp`, `rarp`, `decnet`, `lat`, `sca`, `moprc`, `mopdl`, `tcp` and `udp`. E.g., 'ether src foo', 'arp net 128.3', 'tcp port 21'. If there is no proto qualifier, all protocols consistent with the type are assumed. For example, '`src foo`' means '`(ip or arp or rarp) src foo`' (except the latter is not legal syntax), '`net bar`' means '`(ip or arp or rarp) net bar`' and '`port 53`' means '`(tcp or udp) port 53`'.

['`fddi`' is actually an alias for '`ether`'; the parser treats them identically as meaning "the data link level used on the specified network interface." FDDI headers contain Ethernet-like source and destination addresses, and often contain Ethernet-like packet types, so you can filter on these FDDI fields just as with the analogous Ethernet fields. FDDI headers also contain other fields, but you cannot name them explicitly in a filter expression.]

In addition to the above, there are some special 'primitive' keywords that don't follow the pattern: `gateway`, `broadcast`, `less`, `greater` and arithmetic expressions. All of these are described below.

More complex filter expressions are built up by using the words `and`, `or` and `not` to combine primitives. For example,

```
host foo and not port ftp and not port ftp-data
```

To save typing, identical qualifier lists can be omitted. For example,

`tcp dst port ftp or ftp-data or domain`

is exactly the same as

`tcp dst port ftp or tcp dst port ftp-data or tcp dst \ port domain`

Allowable primitives are:

`dst host` *host* True if the IP destination field of the packet is *host*, which may be either an address or a name.

`src host` *host* True if the IP source field of the packet is *host*.

`host` *host* True if either the IP source or destination of the packet is *host*. Any of the above host expressions can be prepended with the keywords, `ip`, `arp`, or `rarp` as in:

`ip host` *host*

which is equivalent to:

`ether proto` *ip* `and host` *host*

If *host* is a name with multiple IP addresses, each address will be checked for a match.

`ether dst` *ehost*

True if the ethernet destination address is *ehost*. *ehost* may be either a name from `/etc/ethers` or a number (see `ethers`(3N) for numeric format).

`ether src` *ehost*

True if the ethernet source address is *ehost*.

`ether host` *ehost*

True if either the ethernet source or destination address is *ehost*.

`gateway` *host* True if the packet used *host* as a gateway. I.e., the ethernet source or destination address was *host* but neither the IP source nor the IP destination was *host*.

host must be a name and must be found in both `/etc/hosts` and `/etc/ethers`. (An equivalent expression is

`ether host` *ehost* `and not host` *host*

which can be used with either names or numbers for *host* and *ehost*.)

`dst net` *net*
: True if the IP destination address of the packet has a network number of *net*. *net* may be either a name from `/etc/networks` or a network number (see `networks(4)` for details).

`src net` *net*
: True if the IP source address of the packet has a network number of *net*.

`net` *net*
: True if either the IP source or destination address of the packet has a network number of *net*.

`net` *net mask*
: True if the IP address matches *net* with the specific netmask. May be qualified with `src` or `dst`.

`net` *net*/*len*
: True if the IP address matches *net* a netmask *len* bits wide. May be qualified with `src` or `dst`.

`dst port` *port*
: True if the packet is `ip/tcp` or `ip/udp` and has a destination port value of *port*. The *port* can be a number or a name used in `/etc/services` (see `tcp`(4) and `udp`(4)). If a name is used, both the port number and protocol are checked. If a number or ambiguous name is used, only the port number is checked (e.g., `dst port 513` will print both `tcp/login traffic` and `udp/who traffic`, and `port domain` will print both `tcp/domain` and `udp/domain` traffic).

`src port` *port*
: True if the packet has a source port value of *port*.

`port` *port*
: True if either the source or destination port of the packet is *port*. Any of the above port expressions can be prepended with the keywords, `tcp` or `udp`, as in:

`tcp src port` *port*

which matches only `tcp` packets whose source port is *port*.

`less` *length* True if the packet has a length less than or equal to *length*. This is equivalent to:

`len <=` *length*

`greater` *length* True if the packet has a length greater than or equal to *length*. This is equivalent to:

`len >=` *length*

`ip proto` *protocol*
True if the packet is an IP packet (see `ip`(4P)) of protocol type *protocol*. *protocol* can be a number or one of the names `icmp`, `igrp`, `udp`, `nd`, or `tcp`. Note that the identifiers `tcp`, `udp`, and `icmp` are also keywords and must be escaped via backslash (\), which is \\ in the C-shell.

`ether broadcast`
True if the packet is an ethernet broadcast packet. The `ether` keyword is optional.

`ip broadcast` True if the packet is an IP broadcast packet. It checks for both the all-zeroes and all-ones broadcast conventions, and looks up the local subnet mask.

`ether multicast`
True if the packet is an ethernet multicast packet. The `ether` keyword is optional. This is shorthand for

```
ether[0] & 1 != 0
```

`ip multicast` True if the packet is an IP multicast packet.

`ether proto` *protocol*
True if the packet is of ether type *protocol*. *protocol* can be a number or a name like `ip`, `arp`, or `rarp`. Note these identifiers are also keywords and must be escaped via backslash (\). [In the case of FDDI (e.g., '`fddi protocol arp`'), the protocol identification comes from the 802.2 Logical Link Control (LLC) header, which is usually layered on top of the FDDI header. `tcpdump` assumes, when filtering on the protocol identifier, that all FDDI packets include an LLC header, and that the LLC header is in so-called SNAP format.]

`decnet src` *host*
: True if the DECNET source address is *host*, which may be an address of the form "10.123", or a DECNET host name. [DECNET host name support is only available on Ultrix systems that are configured to run DECNET.]

`decnet dst` *host*
: True if the DECNET destination address is *host*.

`decnet host` *host*
: True if either the DECNET source or destination address is *host*.

`ip`, `arp`, `rarp`, `decnet`
: Abbreviations for:

`ether proto` *p* where *p* is one of the above protocols.

`lat`, `moprc`, `mopdl`
: Abbreviations for:

`ether proto` *p* where *p* is one of the above protocols. Note that `tcpdump` does not currently know how to parse these protocols.

`tcp`, `udp`, `icmp` Abbreviations for:

`ip proto` *p* where *p* is one of the above protocols.

expr relop expr True if the relation holds, where *relop* is one of `>`, `<`, `>=`, `<=`, `=`, `!=`, and *expr* is an arithmetic expression composed of integer constants (expressed in standard C syntax), the normal binary operators [`+`, `-`, `*`, `/`, `&`, `|`], a length operator, and special packet data accessors. To access data inside the packet, use the following syntax:

proto `[` *expr* `:` *size* `]`

proto is one of `ether`, `fddi`, `ip`, `arp`, `rarp`, `tcp`, `udp`, or `icmp`, and indicates the protocol layer for the index operation. The byte offset, relative to the indicated protocol layer, is given by *expr*. *size* is optional and indicates the number of bytes in the field of interest; it can be either one, two, or four, and defaults to one. The length operator, indicated by the keyword `len`, gives the length of the packet.

For example, '`ether[0] & 1 != 0`' catches all multicast traffic. The expression '`ip[0] & 0xf != 5`' catches all IP packets with options. The expression '`ip[6:2] & 0x1fff = 0`' catches only unfragmented datagrams and frag zero of fragmented datagrams. This check is implicitly applied to the `tcp` and `udp` index operations. For instance, `tcp[0]` always means the first byte of the TCP *header*, and never means the first byte of an intervening fragment.

Primitives may be combined using:

- A parenthesized group of primitives and operators (parentheses are special to the Shell and must be escaped).
- Negation ('`!`' or '`not`').
- Concatenation ('`&&`' or '`and`').
- Alternation ('`||`' or '`or`').

Negation has highest precedence. Alternation and concatenation have equal precedence and associate left to right. Note that explicit `and` tokens, not juxtaposition, are now required for concatenation.

If an identifier is given without a keyword, the most recent keyword is assumed. For example,

```
not host vs and ace
```

is short for

```
not host vs and host ace
```

which should not be confused with

```
not ( host vs or ace )
```

Expression arguments can be passed to `tcpdump` as either a single argument or as multiple arguments, whichever is more convenient. Generally, if the expression contains Shell metacharacters, it is easier to pass it as a single, quoted argument. Multiple arguments are concatenated with spaces before being parsed.

EXAMPLES

To print all packets arriving at or departing from *sundown*:

```
tcpdump host sundown
```

To print traffic between `helios` and either `hot` or `ace`:

```
tcpdump host helios and \( hot or ace \)
```

To print all IP packets between `ace` and any host except `helios`:

```
tcpdump ip host ace and not helios
```

To print all traffic between local hosts and hosts at Berkeley:

```
tcpdump net ucb-ether
```

To print all ftp traffic through internet gateway `snup`: (note that the expression is quoted to prevent the shell from (mis-)interpreting the parentheses):

```
tcpdump 'gateway snup and (port ftp or ftp-data)'
```

To print traffic neither sourced from nor destined for local hosts (if you gateway to one other net, this stuff should never make it onto your local net).

```
tcpdump ip and not net localnet
```

To print the start and end packets (the SYN and FIN packets) of each TCP conversation that involves a non-local host.

```
tcpdump 'tcp[13] & 3 != 0 and not src and dst net localnet'
```

To print IP packets longer than 576 bytes sent through gateway `snup`:

```
tcpdump 'gateway snup and ip[2:2] > 576'
```

To print IP broadcast or multicast packets that were *not* sent via ethernet broadcast or multicast:

```
tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'
```

To print all ICMP packets that are not echo requests/replies (i.e., not ping packets):

```
tcpdump 'icmp[0] != 8 and icmp[0] != 0"
```

OUTPUT FORMAT

The output of `tcpdump` is protocol dependent. The following gives a brief description and examples of most of the formats.

Link Level Headers

If the **-e** option is given, the link level header is printed out. On ethernets, the source and destination addresses, protocol, and packet length are printed.

On FDDI networks, the **-e** option causes `tcpdump` to print the 'frame control' field, the source and destination addresses, and the packet length. (The frame control field governs the interpretation of the rest of the packet. Normal packets (such as those containing IP datagrams) are async packets, with a priority value between 0 and 7; for example, '`async4`'. Such packets are assumed to contain an 802.2 Logical Link Control (LLC) packet; the LLC header is printed if it is *not* an ISO datagram or a so-called SNAP packet.

(N.B.: The following description assumes familiarity with the SLIP compression algorithm described in RFC-1144.)

On SNAP links, a direction indicator ("`I`" for inbound, "`O`" for outbound), packet type, and compression information are printed out. The packet type is printed first. The three types are `ip`, `utcp`, and `ctcp`. No further link information is printed for `ip` packets. For TCP packets, the connection identifier is printed following the type. If the packet is compressed, its encoded header is printed out. The special cases are printed out as `*S+`*n* and `*SA+`*n*, where *n* is the amount by which the sequence number (or sequence number and ack) has changed. If it is not a special case, zero or more changes are printed. A change is indicated by U (urgent pointer), W (window), A (ack), S (sequence number), and I (packet ID), followed by a delta (+n or -n), or a new value (=n). Finally, the amount of data in the packet and compressed header length are printed.

For example, the following line shows an outbound compressed TCP packet, with an implicit connection identifier; the ack has changed by 6, the sequence number by 49, and the packet ID by 6; there are 3 bytes of data and 6 bytes of compressed header:

```
O ctcp * A+6 S+49 I+6 3 (6)
```

ARP/RARP Packets

Arp/rarp output shows the type of request and its arguments. The format is intended to be self explanatory. Here is a short sample taken from the start of an 'rlogin' from host `rtsg` to host `csam`:

```
arp who-has csam tell rtsg
arp reply csam is-at CSAM
```

The first line says that rtsg sent an arp packet asking for the ethernet address of internet host csam. Csam replies with its ethernet address (in this example, ethernet addresses are in caps and internet addresses in lower case).

This would look less redundant if we had done `tcpdump -n`:

```
arp who-has 128.3.254.6 tell 128.3.254.68
arp reply 128.3.254.6 is-at 02:07:01:00:01:c4
```

If we had done `tcpdump -e`, the fact that the first packet is broadcast and the second is point-to-point would be visible:

```
RTSG Broadcast 0806  64: arp who-has csam tell rtsg
CSAM RTSG 0806  64: arp reply csam is-at CSAM
```

For the first packet this says the ethernet source address is RTSG, the destination is the ethernet broadcast address, the type field contained hex 0806 (type `ETHER_ARP`) and the total length was 64 bytes.

TCP Packets

(N.B.:The following description assumes familiarity with the TCP *protocol described in RFC-793. If you are not familiar with the protocol, neither this description nor tcpdump will be of much use to you.)*

The general format of a tcp protocol line is:

src > dst: flags data-seqno ack window urgent options

src and *dst* are the source and destination IP addresses and ports. *flags* are some combination of `S` (`SYN`), `F` (`FIN`), `P` (`PUSH`) or `R` (`RST`) or a single '.' (no flags). *data-seqno* describes the portion of sequence space covered by the data in this packet (see example below). *ack* is sequence number of the next data expected the other direction on this connection. *window* is the number of bytes of receive buffer space available the other direction on this connection. *urg* indicates there is 'urgent' data in the packet. *options* are tcp options enclosed in angle brackets (e.g., `<mss` 1024>).

src, dst and *flags* are always present. The other fields depend on the contents of the packet's tcp protocol header and are output only if appropriate.

Here is the opening portion of an rlogin from host `rtsg` to host `csam`.

```
rtsg.1023 > csam.login: S 768512:768512(0) win 4096 <mss 1024
> csam.login > rtsg.1023: S 947648:947648(0) ack 768513 win 4096
<mss 1024
> rtsg.1023 > csam.login: . ack 1 win 4096 rtsg.1023
> csam.login: P 1:2(1) ack 1 win 4096 csam.login
> rtsg.1023: . ack 2 win 4096 rtsg.1023
> csam.login: P 2:21(19) ack 1 win 4096
csam.login > rtsg.1023: P 1:2(1) ack 21 win 4077
csam.login > rtsg.1023: P 2:3(1) ack 21 win 4077 urg 1 csam.login
> rtsg.1023: P 3:4(1) ack 21 win 4077 urg 1
```

The first line says that tcp port 1023 on rtsg sent a packet to port `login` on `csam`. The `S` indicates that the `SYN` flag was set. The packet sequence number was 768512 and it contained no data. (The notation is '*first*:*last*(*nbytes*)' which means 'sequence numbers *first* up to but not including *last* which is *nbytes* bytes of user data'.) There was no piggy-backed ack, the available receive window was 4096 bytes and there was a max-segment-size option requesting an mss of 1024 bytes.

The csam system replies with a similar packet except it includes a piggy-backed ack for rtsg's SYN. rtsg then acks csam's SYN. The '.' means no flags were set. The packet contained no data so there is no data sequence number. Note that the ack sequence number is a small integer (1). The first time tcpdump sees a tcp 'conversation', it prints the sequence number from the packet. On subsequent packets of the conversation, the difference between the current packet's sequence number and this initial sequence number is printed. This means that sequence numbers after the first can be interpreted as relative byte positions in the conversation's data stream (with the first data byte each direction being '1'). '**-S**' will override this feature, causing the original sequence numbers to be output.

On the 6th line, rtsg sends csam 19 bytes of data (bytes 2 through 20 in the rtsg → csam side of the conversation). The PUSH flag is set in the packet. On the 7th line, csam says its received data sent by rtsg up to but not including byte 21. Most of this data is apparently sitting in the socket buffer since the receive window for csam has gotten 19 bytes smaller. csam also sends one byte of data to rtsg in this packet. On the 8th and 9th lines, csam sends two bytes of urgent, pushed data to rtsg.

UDP Packets

UDP format is illustrated by this rwho packet:

```
actinide.who > broadcast.who: udp 84
```

This says that port who on host actinide sent a udp datagram to port who on host broadcast, the Internet broadcast address. The packet contained 84 bytes of user data.

Some UDP services are recognized (from the source or destination port number) and the higher level protocol information printed. In particular, Domain Name service requests (RFC-1034/1035) and Sun RPC calls (RFC-1050) to NFS.

UDP Name Server Requests

(N.B.:The following description assumes familiarity with the Domain Service protocol described in RFC-1035. If you are not familiar with the protocol, the following description will appear to be written in greek.)

Name server requests are formatted as

src > dst: id op? flags qtype qclass name (len)

```
h2opolo.1538 > helios.domain: 3+ A? ucbvax.berkeley.edu. (37)
```

Host `h2opolo` asked the domain server on `helios` for an address record (qtype=`A`) associated with the name `ucbvax.berkeley.edu.` The query id was '`3`'. The '+' indicates the *recursion desired* flag was set. The query length was 37 bytes, not including the UDP and IP protocol headers. The query operation was the normal one, *query*, so the *op* field was omitted. If the *op* had been anything else, it would have been printed between the '3' and the '+'. Similarly, the *qclass* was the normal one, `C_IN`, and omitted. Any other qclass would have been printed immediately after the 'A'.

A few anomalies are checked and may result in extra fields enclosed in square brackets: If a query contains an answer, name server or authority section, *ancount*, *nscount*, or *arcount* are printed as '[*n*a]', '[*n*n]' or '[*n*au]' where *n* is the appropriate count. If any of the response bits are set (`AA`, `RA` or rcode) or any of the 'must be zero' bits are set in bytes two and three, '[b2&3=*x*]' is printed, where *x* is the hex value of header bytes two and three.

UDP Name Server Responses

Name server responses are formatted as

src > dst: id op rcode flags a/n/au type class data (len)

```
helios.domain > h2opolo.1538: 3 3/3/7 A 128.32.137.3 (273)
helios.domain > h2opolo.1537: 2 NXDomain* 0/1/0 (97)
```

In the first example, `helios` responds to query id 3 from `h2opolo` with 3 answer records, 3 name server records and 7 authority records. The first answer record is type `A` (address) and its data is internet address 128.32.137.3. The total size of the response was 273 bytes, excluding UDP and IP headers. The op (Query) and response code (NoError) were omitted, as was the class (`C_IN`) of the `A` record.

In the second example, `helios` responds to query 2 with a response code of non-existent domain (`NXDomain`) with no answers, one name server and no authority records. The '*' indicates that the *authoritative answer* bit was set. Since there were no answers, no type, class or data were printed.

Other flag characters that might appear are '**–**' (recursion available, `RA`, *not* set) and '**|**' (truncated message, `TC`, set). If the 'question' section doesn't contain exactly one entry, '[*n*q]' is printed.

Note that name server requests and responses tend to be large and the default *snaplen* of 68 bytes may not capture enough of the packet to print. Use the **-s** flag to increase the snaplen if you need to seriously investigate name server traffic. '`-s 128`' has worked well for me.

NFS Requests and Replies

Sun NFS (Network File System) requests and replies are printed as:

src.xid > dst.nfs: len op args
src.nfs > dst.xid: reply stat len op results

```
sushi.6709 > wrl.nfs: 112 readlink fh 21,24/10.73165
wrl.nfs > sushi.6709: reply ok 40 readlink "../var"
sushi.201b > wrl.nfs:
  144 lookup fh 9,74/4096.6878 "xcolors"
wrl.nfs > sushi.201b:
  reply ok 128 lookup fh 9,74/4134.3150
```

In the first line, host `sushi` sends a transaction with id `6709` to `wrl` (note that the number following the src host is a transaction id, *not* the source port). The request was 112 bytes, excluding the UDP and IP headers. The operation was a `readlink` (read symbolic link) on file handle (`fh`) 21,24/10.731657119. (If one is lucky, as in this case, the file handle can be interpreted as a major,minor device number pair, followed by the inode number and generation number.) `wrl` replies '`ok`' with the contents of the link.

In the third line, `sushi` asks `wrl` to lookup the name '`xcolors`' in directory file 9,74/4096.6878. Note that the data printed depends on the operation type. The format is intended to be self explanatory if read in conjunction with an NFS protocol spec.

If the **-v** (verbose) flag is given, additional information is printed. For example:

```
sushi.1372a > wrl.nfs:
  148 read fh 21,11/12.195 8192 bytes @ 24576
wrl.nfs > sushi.1372a:
  reply ok 1472 read REG 100664 ids 417/0 sz 29388
```

(**-v** also prints the IP header TTL, ID, and fragmentation fields, which have been omitted from this example.) In the first line, `sushi` asks `wrl` to read 8192 bytes from file 21,11/12.195, at byte offset 24576. `Wrl` replies 'ok'; the packet shown on the second line is the first fragment of the reply, and hence is only 1472 bytes long (the other bytes will follow in subsequent fragments, but these fragments do not have NFS or even UDP headers and so might not be printed, depending on the filter expression used). Because the **-v** flag is given, some of the file attributes (which are returned in addition to the file data) are printed: the file type ("`REG`", for regular file), the file mode (in octal), the uid and gid, and the file size.

If the **-v** flag is given more than once, even more details are printed.

Note that NFS requests are very large and much of the detail won't be printed unless *snaplen* is increased. Try using '`-s 192`' to watch NFS traffic.

NFS reply packets do not explicitly identify the RPC operation. Instead, `tcpdump` keeps track of "recent" requests, and matches them to the replies using the transaction ID. If a reply does not closely follow the corresponding request, it might not be parsable.

KIP Appletalk (DDP in UDP)

Appletalk DDP packets encapsulated in UDP datagrams are de-encapsulated and dumped as DDP packets (i.e., all the DDP header information is discarded). The file `/etc/atalk.names` is used to translate appletalk net and node numbers to names. Lines in this file have the form

number	*name*
`1.254`	`ether`
`16.1`	`icsd-net`

```
1.254.110      ace
```

The first two lines give the names of appletalk networks. The third line gives the name of a particular host (a host is distinguished from a net by the 3rd octet in the number – a net number *must* have two octets and a host number *must* have three octets.) The number and name should be separated by whitespace (blanks or tabs). The `/etc/atalk.names` file may contain blank lines or comment lines (lines starting with a '#').

Appletalk addresses are printed in the form

net.host.port

```
144.1.209.2 > icsd-net.112.220
office.2 > icsd-net.112.220
jssmag.149.235 > icsd-net.2
```

(If the `/etc/atalk.names` doesn't exist or doesn't contain an entry for some appletalk host/net number, addresses are printed in numeric form.) In the first example, NBP (DDP port 2) on net 144.1 node 209 is sending to whatever is listening on port 220 of net icsd node 112. The second line is the same except the full name of the source node is known ('office'). The third line is a send from port 235 on net jssmag node 149 to broadcast on the icsd-net NBP port (note that the broadcast address (255) is indicated by a net name with no host number – for this reason it's a good idea to keep node names and net names distinct in `/etc/atalk.names`).

NBP (name binding protocol) and ATP (Appletalk transaction protocol) packets have their contents interpreted. Other protocols just dump the protocol name (or number if no name is registered for the protocol) and packet size.

NBP `packets` are formatted like the following examples:

```
icsd-net.112.220 > jssmag.2: nbp-lkup 190: "=:LaserWriter@*"
jssmag.209.2 > icsd-net.112.220: nbp-reply 190:
"RM1140:LaserWriter@*" 250
techpit.2 > icsd-net.112.220: nbp-reply 190:
"techpit:LaserWriter@*" 186
```

The first line (`icsd-net` through `nbp-reply 190:`) is a name lookup request for laserwriters sent by net icsd host 112 and broadcast on net jssmag. The nbp id for the lookup is 190. The second line shows a reply for this request (note that it has the same id) from host jssmag.209 saying that it has a laserwriter resource named "RM1140" registered on port 250. The third line (techpit.2 and beyond) is another reply to the same request saying host techpit has laserwriter "techpit" registered on port 186.

ATP packet formatting is demonstrated by the following example:

```
jssmag.209.165 > helios.132: atp-req  12266<0-7 > 0xae030001
helios.132 > jssmag.209.165: atp-resp 12266:0 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:1 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:2 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:3 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:4 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:5 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:6 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp*12266:7 (512) 0xae040000
jssmag.209.165 > helios.132: atp-req  12266<3,5> 0xae030001
helios.132 > jssmag.209.165: atp-resp 12266:3 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:5 (512) 0xae040000
jssmag.209.165 > helios.132: atp-rel  12266<0-7> 0xae030001
jssmag.209.133 > helios.132: atp-req* 12267<0-7> 0xae030002
```

jssmag.209 initiates transaction id 12266 with host helios by requesting up to 8 packets (the '<0-7>'). The hex number at the end of the line is the value of the 'userdata' field in the request.

helios responds with 8 512-byte packets. The ':*digit*' following the transaction id gives the packet sequence number in the transaction and the number in parens is the amount of data in the packet, excluding the atp header. The '*' on packet 7 indicates that the EOM bit was set.

jssmag.209 then requests that packets 3 & 5 be retransmitted. helios resends them then jssmag.209 releases the transaction. Finally, jssmag.209 initiates the next request. The '*' on the request indicates that XO ('exactly once') was *not* set.

IP Fragmentation

Fragmented Internet datagrams are printed as

(frag *id*:*size*@*offset*+)
(frag *id*:*size*@*offset*)

(The first form indicates there are more fragments. The second indicates this is the last fragment.)

Id is the fragment id. *size* is the fragment size (in bytes) excluding the IP header. *Offset* is this fragment's offset (in bytes) in the original datagram.

The fragment information is output for each fragment. The first fragment contains the higher level protocol header and the frag info is printed after the protocol info. Fragments after the first contain no higher level protocol header and the frag info is printed after the source and destination addresses. For example, here is part of an ftp from `arizona.edu` to `rtsg.arpa` over a CSNET connection that doesn't appear to handle 576 byte datagrams:

```
arizona.ftp-data > rtsg.1170: . 1024:1332(308) ack 1 win 4096
(frag 595a:328@0+)
arizona > rtsg: (frag 595a:204@328)
rtsg.1170 > arizona.ftp-data: . ack 1536 win 2560
```

There are a couple of things to note here: First, addresses in the 2nd line don't include port numbers. This is because the TCP protocol information is all in the first fragment and we have no idea what the port or sequence numbers are when we print the later fragments. Second, the tcp sequence information in the first line is printed as if there were 308 bytes of user data when, in fact, there are 512 bytes (308 in the first frag and 204 in the second). If you are looking for holes in the sequence space or trying to match up acks with packets, this can fool you.

A packet with the IP *don't fragment* flag is marked with a trailing `(DF)`.

Timestamps

By default, all output lines are preceded by a timestamp. The timestamp is the current clock time in the form

hh:mm:ss.frac

and is as accurate as the kernel's clock. The timestamp reflects the time the kernel first saw the packet. No attempt is made to account for the time lag between when the ethernet interface removed the packet from the wire and when the kernel serviced the 'new packet' interrupt.

SEE ALSO

Device Facilities - `bpf`

AUTHORS

Van Jacobson (`van@ee.lbl.gov`), Craig Leres (`leres@ee.lbl.gov`) and Steven McCanne (`mccanne@ee.lbl.gov`), all of the Lawrence Berkeley Laboratory, University of California, Berkeley, CA.

BUGS

Please send bug reports to `tcpdump@ee.lbl.gov` or `libpcap@ee.lbl.gov`.

NIT doesn't let you watch your own outbound traffic, BPF will. We recommend that you use the latter.

`tcpdump` for Ultrix requires Ultrix version 4.0 or later; the kernel has to have been built with the `packetfilter` pseudo-device driver (see `packetfilter`(4)). In order to watch either your own outbound or inbound traffic, you will need to use Ultrix version 4.2 or later, and you will have to have used the `pfconfig`(8) command to enable "copyall" mode.

Under SunOS 4.1, the packet capture code (or Streams NIT) is not what you'd call efficient. Don't plan on doing much with your Sun while you're monitoring a busy network.

On Sun systems prior to release 3.2, NIT is very buggy. If run on an old system, tcpdump may crash the machine.

Some attempt should be made to reassemble IP fragments or, at least to compute the right length for the higher level protocol.

Name server inverse queries are not dumped correctly: The (empty) question section is printed rather than real query in the answer section. Some believe that inverse queries are themselves a bug and prefer to fix the program generating them rather than `tcpdump`.

Apple Ethertalk DDP packets could be dumped as easily as KIP DDP packets but aren't. Even if we were inclined to do anything to promote the use of Ethertalk (we aren't), LBL doesn't allow Ethertalk on any of its networks so we'd would have no way of testing this code.

A packet trace that crosses a daylight savings time change will give skewed time stamps (the time change is ignored).

Filters expressions that manipulate FDDI headers assume that all FDDI packets are encapsulated Ethernet packets. This is true for IP, ARP, and DECNET Phase IV, but is not true for protocols such as ISO CLNS. Therefore, the filter may inadvertently accept certain packets that do not properly match the filter expression.

NAME

`tcsh` – command interpreter with C-like syntax

SYNOPSIS

`tcsh` **[-bcdefimnqstvVxX] [** *argument* ... **]**

`tcsh` **-l** *OS/Dependent Options*

`tcsh` **[-F] [-D***name***[=***value***]]**

DESCRIPTION

`tsch` is an enhanced version of the Berkeley UNIX C shell `csh`. It behaves exactly like the C shell, except for the added utilities of:

1. Command line editing using Emacs-style commands.
2. Visual step up/down through the history list.
3. Terminal mode sanity checking and resetting.
4. Interactive command, file name, and user name completion.
5. `File/directory/user` list in the middle of a typed command.
6. Spelling correction of command, file, and user names.
7. Lookup of command documentation in the middle of a typed command.
8. Enhanced history mechanism.
9. Automatic locking/logout after long periods of idle time.
10. Automatic execution of a single command prior to printing each prompt.
11. Automatic periodic command execution.
12. A new syntax for the prompt, and the ability to set the prompt for `while` and `for` loops.
13. Time stamps in the history list.
14. An addition to the syntax of filenames to access entries in the directory stack, and the ability treat symbolic links in a sane way when changing directories.

15. The ability to watch for logins and logouts by user or terminal on the machine.
16. A scheduled event list, which specifies commands which are to be executed at given times.
17. A new builtin that does a subset of `ls`.
18. An addition to the file expression syntax for a character not in a set of characters and the ability to negate a globbing pattern.
19. New automatically initialized environment variables *HOST* and *HOSTTYPE*.
20. Commands for debugging terminal capabilities.
21. Searching for the visual history mechanism.
22. New builtins for the `which` and `where` commands.
23. Restarting a stopped editor with two keystrokes.
24. Status line support
25. Automatic execution of a command when the current working directory is changed.
26. Native language system support.
27. Automatic process time reporting.
28. OS-dependent builtin support.
29. Automatic window size adjustment.
30. Input files.
31. Additional/undocumented options.
32. Enhanced history/variable modifier expansion.
33. Programmable `file/user/variable/alias` completion.

For a description of standard C-shell features, see the `csh` manual page.

Command Line Editing

Commands that the user types may be edited using the same control characters that GNU Emacs, or `vi` uses. Arrow and function key sequences are also allowed. `tcsh` allows this by setting the terminal to `CBREAK` mode and reading the input one character at a time.

There is a new shell `bindkey` that allows the user to redefine what any key does, or find out what any or all of the keys do. The syntac is:

bindkey [-a] [-c] [-s] [-v] [-e] [-d] [-l] [-r] [--] /
[*in-string* **[***out-string* **|** *command***]]**

If no values are given all bindings are listed. If only *in-string* is given, bindings for the *in-string* is listed.

Otherwise it binds the *in-string* to the given *out-string* or command. If *out-string*, this is treated as input to `tcsh` when *in-string* is typed. This may be used recursively to currently a level of 10 deep.

There are two basic key maps: the normal and alternative one. The alternative is used by vi command mode. For multi-character input, the basic key maps contains a sequence-lead-in for the first character in the input.

Options:

`-a`	Bind *in-string* in alternative key map
`-s`	Bind an *out-string* instead of a command
`-c`	Bind a UNIX command instead of a command
`-v`	Bind for default `vi`-mode
`-e`	Bind for default emacs-mode
`-d`	Bind to compiled in default
`-l`	List functions available with short description
`-r`	Remove the binding of *in-string*

In strings control characters may be written as `caret-<letter>`, and backslash (\) is used to escape a character as follows:

`\a`	Bell character
`\n`	Line feed (new line)
`\b`	Back space

`\t`	Horizontal tab
`\v`	Vertical tab
`\f`	Form feed
`\r`	Carriage return
`\e`	Escape
`\nnn`	Character code in octal

In all other cases `\` escapes the following character. Needed for escaping the special meaning of `\` and `^`. Delete is written as `^?` (caret-question mark).

`tsch` always binds the arrow keys as defined by the `termcap` entry to:

`up arrow`	up-history
`down arrow`	down-history
`right arrow`	forward-char
`left arrow`	backward-char

Except where these bindings would alter other single character bindings. If this is not desired one can avoid the automatic arrow key binding using `settc` to change the arrow key escape sequences to the empty string. The `ansi/vt100` sequences for arrow keys are always bound.

Presented within the table "Emacs Bindings" and the table "Emacs Multi-Character & 8-Bit Bindings" are lists of the default `emacs` bindings.

Presented within the table "vi Insert Mode Functions", the table "vi Command Mode Functions", and the table "vi Multi-Character Bindings" are lists of the default `vi` bindings.

Characters with the 8th bit set are written as `M-<character>`. Note however, that unlike with the old `bind` command (see below), the syntax `M-<character>` has no special meaning to the `bindkey` command, and the bindings for the sequence `escape+<character>` and `M-<character>` as given below are handled separately (although the default bindings are the same). The printable ASCII characters not mentioned in the list are bound to the `self-insert-command` function, which just inserts the given character literally into the input line.

The remaining characters are bound to the `undefined-key` function, which only causes a beep (unless `nobeep` is set, of course).

Table 1: Emacs Bindings

Bindings	Description	Bindings	Description
`^@`	set-mark-command	`^Q`	tty-start-output
`^A`	beginning-of-line	`^R`	redisplay
`^B`	backward-char	`^S`	tty-stop-output
`^C`	tty-sigintr	`^T`	transpose-chars
`^D`	delete-char-or-list	`^U`	kill-whole-line
`^E`	end-of-line	`^V`	quoted-insert
`^F`	forward-char	`^W`	kill-region
`^G`	is undefined	`^X`	sequence-lead-in
`^H`	backward-delete-char	`^Y`	yank
`^I`	complete-word	`^Z`	tty-sigtsusp
`^J`	newline	`^[`	sequence-lead-in
`^K`	kill-line	`^\`	tty-sigquit
`^L`	clear-screen	`^]`	tty-dsusp
`^M`	newline	“ ” `to /`	self-insert-command
`^N`	down-history	`0 to 9`	digit
`^O`	tty-flush-output	`: to ~`	self-insert-command
`^P`	up-history	`^?`	backward-delete-char

Table 2: Emacs Multi-Character & 8-Bit Bindings

8-bit Binding	Description	8-bit Binding	Description
`^[^D or M-^D`	list-choices	`^[N or M-N`	history-search-forward
`^[^H or M-^H`	backward-delete-word	`^[P or M-P`	history-search-backward
`^[^I or M-^I`	complete-word	`^[R or M-R`	toggle-literal-history
`^[^L or M-^L`	clear-screen	`^[S or M-S`	spell-word
`^[^Z or M-^Z`	run-`fg`-editor	`^[U or M-U`	uppercase-word
`^[^[ or M-^[`	complete-word	`^[W or M-W`	copy-region-as-kill
`^[^_ or M-^_`	copy-prev-word	`^[_ or M-_`	insert-last-word
`^[ or M-`	expand-history	`^[b or M-b`	backward-word
`^[! or M-!`	expand-history	`^[c or M-c`	capitalize-word
`^[$ or M-$`	spell-line	`^[d or M-d`	delete-word
`^[0 or M-0`	digit-argument	`^[f or M-f`	forward-word
`^[1 or M-1`	digit-argument	`^[h or M-h`	run-help
`^[2 or M-2`	digit-argument	`^[l or M-l`	downcase-word
`^[3 or M-3`	digit-argument	`^[n or M-n`	history-search-forward
`^[4 or M-4`	digit-argument	`^[p or M-p`	history-search-backward
`^[5 or M-5`	digit-argument	`^[r or M-r`	toggle-literal-history
`^[6 or M-6`	digit-argument	`^[s or M-s`	spell-word
`^[7 or M-7`	digit-argument	`^[u or M-u`	uppercase-word
`^[8 or M-8`	digit-argument	`^[w or M-w`	copy-region-as-kill

Table 2: Emacs Multi-Character & 8-Bit Bindings

8-bit Binding	Description	8-bit Binding	Description
`^[9 or M-9`	digit-argument	`^[^? or M-^?`	backward-delete-word
`^[? or M-?`	which-command	`^X^X`	exchange-point-and-mark
`^[B or M-B`	backward-word	`^X*`	expand-glob
`^[C or M-C`	capitalize-word	`^X$`	expand-variables
`^[D or M-D`	delete-word	`^XG`	list-glob
`^[F or M-F`	forward-word	`^Xg`	list-glob
`^[H or M-H`	run-help	`^Xn`	normalize-path
`^[L or M-L`	downcase-word	`^XN`	normalize-path

Table 3: vi Insert Mode Functions

Insert Mode	Description	Insert Mode	Description	Insert Mode	Description
`^@`	is undefined	`^K`	kill-line	`^U`	backward-kill-line
`^A`	beginning-of-line	`^L`	clear-screen	`^V`	quoted-insert
`^B`	backward-char	`^M`	newline	`^W`	backward-delete-word
`^C`	`tty-sigintr`	`^N`	down-history	`^X`	expand-line
`^D`	list-or-eof	`^O`	`tty`-flush-output	`^Y`	`tty-dsusp`
`^E`	end-of-line	`^P`	up-history	`^Z`	`tty-sigtsusp`
`^F`	forward-char	`^Q`	`tty`-start-output	`^[`	vi-`cmd`-mode
`^G`	list-glob	`^R`	redisplay	`^\`	`tty-sigquit`
`^H`	backward-delete-char	`^S`	`tty`-stop-output	“ ” `to ~`	self-insert-command
`^I`	complete-word	`^T`	transpose-chars	`^?`	backward-delete-char
`^J`	newline				

Table 4: vi Command Mode Functions

vi	Description	vi	Description	vi	Description
^@	is undefined	$	end-of-line	X	backward-delete-char
^A	beginning-of-line	*	expand-glob	[	sequence-lead-in
^B	is undefined	+	down-history	\^	beginning-of-line
^C	tty-sigintr	;	vi-repeat-char-fwd	a	vi-add
^D	list-choices	,	vi-repeat-char-back	b	backward-word
^E	end-of-line	–	up-history	c	is undefined
^F	is undefined	.	is undefined	d	vi-delmeta
^G	list-glob	/	vi-search-back	e	vi-eword
^H	backward-char	0	vi-zero	f	vi-char-fwd
^I	vi-cmd-mode-complete	1 to 9	digit-argument	h	backward-char
^J	newline	?	vi-search-fwd	i	vi-insert
^K	kill-line	@	is undefined	j	down-history
^L	clear-screen	A	vi-add-at-eol	k	up-history
^M	newline	B	vi-word-back	l	forward-char
^N	down-history	C	change-till-end-of-line	m	is undefined
^O	tty-flush-output	D	kill-line	n	vi-search-fwd
^P	up-history	E	vi-endword	r	vi-replace-char
^Q	tty-start-output	F	vi-char-back	s	vi-substitute-char
^R	redisplay	I	vi-insert-at-bol	t	vi-charto-fwd
^S	tty-stop-output	J	history-search-forward	u	vi-undo

Table 4: vi Command Mode Functions

vi	Description	vi	Description	vi	Description
`^T`	is undefined	`K`	history-search-backward	`v`	expand-variables
`^U`	backward-kill-line	`N`	vi-search-back	`w`	vi-beginning-of-next-word
`^V`	is undefined	`O`	sequence-lead-in	`x`	delete-char
`^W`	backward-delete-word	`R`	vi-replace-mode	`~`	change-case
`^X`	expand-line	`S`	vi-substitute-line	`^?`	backward-delete-char
`^[`	sequence-lead-in	`T`	vi-charto-back	`M-?`	run-help
`^\`	`tty-sigquit`	`U`	is undefined	`M-[`	sequence-lead-in
“ ”	forward-char	`V`	expand-variables	`M-O`	sequence-lead-in
`!`	expand-history	`W`	vi-word-fwd		

Table 5: vi Multi-Character Bindings

vi	Description
`^[?`	run-help

In `tcsh` there are two input modes: `insert` and `overwrite`. All characters that are bound to the `self-insert-command` either get inserted or overwrite what was under the cursor depending on the input mode. In emacs one can toggle between modes using the `overwrite-mode` function, and in `vi` one can use any of the replace mode commands. By default the mode is preserved between input lines. To specify a default mode that can be enforced at the beginning of each command line, the variable `inputmode` can be set to either `insert` or `overwrite`.

There is also an older version of `bindkey` called `bind` that allows the user to redefine what any key does, or find out what any or all of the keys do. This is retained for compatibility reasons.

If given two arguments, `bind` binds the function (first argument) to the given key (second argument). The key may be: the direct character or a `caret-<letter>` combination, which is converted to `control-<letter>`; `M-<letter>` for an escaped character; or `F-<string>` for a function key. For the last of these, the function key prefix must be bound to the function sequence-lead-in and the string specified to the `bind` command must not include this prefix.

If given one argument, `bind` takes the argument as the name for a key and tells what that key does. As a special case, the user can say

```
bind emacs
```

or

```
bind vi
```

to bind all the keys for Emacs or `vi` mode respectively.

If given no arguments, `bind` tells what all of the keys do. If you give `bind` the single argument of defaults, it resets each key to its default value (see the above list).

Visual History

The keys `^P` and `^N` are used to step up and down the history list. If the user has typed in the following:

```
> ls
foo    bar
> echo mumble
mumble
>
```

then enters `^P,` the shell will place `echo mumble` into the editing buffer, and will put the cursor at the end of the line. If another `^P` is entered, then the editing line will change to `ls`. `More ^As` will make the bell ring, since there are no more lines in the history. `^N` works the same way, except it steps down (forward in time).

An easy way to re-do a command is to type `^P` followed by `Return`. Also, pieces of previous commands can be assembled to make a new command. The commands that work on regions are especially useful for this.

`^P` and `^N` actually only copy commands from out of the history into the edit buffer; thus the user may step back into the history and then edit things, but those changes do not affect what is actually in `tcsh`'s history.

Another way to recall (parts of) history commands is via the expand-history function. A variation of the expand-history function is called magic-space. This function expands `csh` history, and always appends a space. Magic-space thus can be bound to `<space>`, to automatically expand `csh` history. Expand-history is normally bound to `M-<space>` and magic-space is not bound.

tty Mode Sanity

As part of the editor, `tcsh` does a check and reset of the terminal mode bits. If the speed has been changed, then `tcsh` will change to using that speed. `tsch` will also obey changes in the padding needed by the `tty`. Also, the shell will automatically turn off `RAW` and `CBREAK` (on systems that use `termio` it will turn on `ICANON`) modes, and will turn on the `tty` driver's output processing (`OPOST`).

The list of the `tty` modes that are always set or cleared by `tcsh` can be examined and modified using the `setty` builtin. The `setty` display is similar to `stty,` and varies depending on the system's `tty` driver. Modes that `tcsh` will always try to set are shown as `+mode`. Modes that `tcsh` will always try to clear are shown as `-mode` and modes that `tcsh` will track and allow to be modified are not shown by default, or if the `-a` flag is given, are shown without a leading sign. For `tty` characters, `-char` disables the `tty` character, while `+char` protects the `tty` character from being changed.

`setty` can be used to control the three different modes `tcsh` uses for the `tty`. The default mode displayed is the `execute` mode which is used when `tcsh` executes commands. The two other modes are the `edit` mode and the `quote` mode. The `edit` mode is used for command line editing, and the `quote` mode is used when entering literal characters.

The `execute` mode can be specified with `-x`, the `edit` mode with `-d`, and the `quote` mode with `-q`. For example, if one wants to set the `echok` flag and let the `echoe` pass unchanged:

```
> setty
iflag:-inlcr -igncr +icrnl
oflag:+opost +onlcr -onlret
cflag:
lflag:+isig +icanon +echo +echoe -echok -echonl -noflsh /
  +echoctl -flusho +iexten
chars:
> setty +echok echoe
> setty
iflag:-inlcr -igncr +icrnl
oflag:+opost +onlcr -onlret
cflag:
lflag:+isig +icanon +echo +echok -echonl -noflsh +echoctl
  -flusho +iexten
chars:
```

Word Completion

In typing commands, it is no longer necessary to type a complete name, only a unique abbreviation is necessary. When you type a `TAB` to `tcsh` it will complete the name for you, echoing the full name on the terminal (and entering it into the edit buffer). If the prefix you typed matches no name, the terminal bell is rung, unless the variable `nobeep` is set. The name may be partially completed if the prefix matches several longer names. If this is the case, the name is extended up to the point of ambiguity, and the bell is rung. This works for file names, command names, shell variables and the ~ user name convention. The variable `fignore` may be set to a list of suffixes to be disregarded during completion.

Example—assume the current directory contained the files:

```
DSC.TXT    bin     cmd       lib       memos
DSC.NEW    chaos   cmtest    mail      netnews
bench      class   dev       mbox      new
```

The command:

```
> gnumacs ch[TAB]
```

would cause `tcsh` to complete the command with the file name `chaos`. If instead, the user had typed:

```
> gnumacs D[TAB]
```

`tcsh` would have extended the name to `DSC` and rung the terminal bell, indicating partial completion. However, if `fignore` had previously been set to a list containing `.NEW` as one element; e.g., (`.o .NEW`), `tcsh` would have completed the `D` to `DSC.TXT`.

File name completion works equally well when other directories are addressed. Additionally, `tcsh` understands the C shell tilde (~) convention for home directories. In addition it caches tilde name references for fast access. This cache can be flushed by using the builtin `rehash.` Thus,

```
> cd ~speech/data/fr[TAB]
```

does what one might expect. This may also be used to expand login names only. Thus,

```
> cd ~sy[TAB]
```

expands to

```
> cd ~synthesis
```

Command names may also be completed, for example,

```
> gnum[TAB]
```

will expand to `gnumacs` (assuming that there are no other commands that begin with `gnum`). Shell and environment variables are recognized also and in addition they can be expanded:

```
> set local=/usr/local
> echo $lo[TAB]
```

will expand to `$local/`. Note that a slash is appended because the expanded variable points to a directory. Also:

```
> set local=/usr/local
> echo $local/[^D]
bin/ etc/ lib/ man/ src/
```

will correctly list the contents of `/usr/local`. Shell and environment variables can also be expanded via the expand-variables function:

```
> echo $local/[^X$]
> echo /usr/local/
```

Completion also works when the cursor is in the middle of the line, rather than just the end. All of the text after the cursor will be saved, the completion will work (possibly adding to the current name), and then the saved text will be restored in place, after the cursor.

The behavior of the completion can be changed by the setting of several shell variables:

Setting the `recexact` variable makes an exact command be expanded rather than just ringing the bell. For example, assume the current directory has two sub-directories called foo and food, then with `recexact` set the following could be done:

```
> cd fo[TAB]
```

to ...

```
> cd foo[TAB]
```

to ...

```
> cd foo/
```

rather than beeping on the second `TAB`.

If the `autolist` variable is set, invoking completion when several choices are possible will automatically list the choices, effectively merging the functionality described in the next section into the completion mechanism. If `autolist` is set to `ambiguous`, then `tcsh` will only list the choices if there were no new characters added in the completion operation.

The noise level can be controlled by the value that `matchbeep` is set to

- `matchbeep=nomatch`, completion will only beep if there are no matching names

- `matchbeep=ambiguous`, completion will `also` beep if there are many possible matches
- `matchbeep=notunique`, completion will `also` beep when there is an exact match but there are other, longer, matches (see `recexact`).
- `matchbeep=never` or set to any other value, completion will never beep. If `matchbeep` is not set it defaults to ambiguous.

If the `autoexpand` variable is set, the expand-history function will be invoked automatically before the completion attempt, expanding normal `csh` history substitutions.

For covert operation, the variable `nobeep` can be set; it will prevent the completion mechanism, as well as `tcsh` in general, from actually beeping. Finally, if the `autocorrect` variable is set, the spelling correction is attempted for any path components up to the completion point.

Listing Of Possible Names

At any point in typing a command, you may request “what names are available”. Thus, when you have typed, perhaps:

```
> cd ~speech/data/fritz/
```

you may wish to know what files or sub-directories exist (in `~speech/data/fritz`), without, of course, aborting the command you are typing. Typing the character Control-D (`^D`), will list the names (files, in this case) available. The files are listed in multi-column format, sorted column-wise. Directories are indicated with a trailing `/`, executable files with a `*`, symbolic links with a `@`, sockets with a `=`, FIFOs (named pipes) with a `|`, character devices with a `%`, and block devices with a `#`. Once printed, the command is re-echoed for you to complete.

Additionally, one may want to know which files match a prefix. If the user had typed:

```
> cd ~speech/data/fr[^D]
```

all files and sub-directories whose prefix was `fr` would be printed. Notice that the example before was simply a degenerate case of this with a null trailing file name. (The null string is a prefix of all strings.) Notice also, that a trailing slash is required to pass to a new directory for both file name completion and listing.

The degenerate

```
> ~[^D]
```

will print a full list of login names on the current system. Note, however, that the degenerate

```
> <Spaces>[^D]
```

does not list all of the commands, but only the files in the current working directory.

Listing/expanding of words that match a name containing wildcard characters can be done via the list-glob/expand-glob function:

```
> ls
foo.c bar.c a.out
> vi *.c[^Xg]
foo.c bar.c
> vi *.c[^X*]
> vi foo.c bar.c
```

Command Name Recognition

Command name recognition and completion works in the same manner as file name recognition and completion above. The current value of the environment variable `PATH` is used in searching for the command. For example

```
> newa[TAB]
```

might expand to

```
> newaliases
```

Also,

```
> new[^D]
```

would list all commands (along `PATH`) that begin with `new`.

Note that `Control-D` has three different effects on `tcsh`. On an empty line (one that contains nothing, not even spaces), `^D` sends an `EOF` to `tcsh` just as it does for normal programs. When the cursor is in the middle of a line of text, `^D` deletes the character that the cursor is under. Finally, a `^D` at the end of a line of text lists the available names at that point. To get a list of available names when the cursor is in the middle of a line (or on an empty line), a `Meta-Control-D` should be typed (`Escape` followed by `Control-D`).

Spelling Correction

If while typing a command, the user mistypes or misspells a file name, user name, or command name, `tcsh` can correct the spelling. When correcting a file name, each part of the path is individually checked and corrected. Spelling correction can be invoked in several different ways:

The `spell-word` function, normally bound to `M-s` (and `M-S`), will attempt to correct the word immediately before the cursor. For example, suppose that the user has typed:

```
> cd /uxr/spol/news[ESC s]
```

`tsch` will check the path for spelling, correct the mistakes, and redraw the line as

```
> cd /usr/spool/news
```

leaving the cursor at the end of the line.

Spelling correction of the entire command line (independent of where the cursor is) can be done with the `spell-line` function, normally bound to `M-$` (Escape Dollar-sign). It will check each word independently, but in order to avoid e.g. command options, no correction is attempted on words whose first character is found in the string `!.\^-*%`.

Finally, automatic spelling correction will be done each time the Return key is hit, if the `correct` variable is set to an appropriate value: `correct=cmd` will cause the spelling of the command name only to be checked, while `correct=all` causes checking of all words on the line, like the spell-line function. If any part of the command line is corrected, the user will be given a special prompt as defined by the `prompt3` variable, followed by the corrected line, e.g.

```
> lz /usr/bin
CORRECT>ls /usr/bin (y|n|e)?
```

Answering `y` or `<space>` at the prompt will cause the corrected line to be executed, `e` will leave the uncorrected command in the input buffer and anything else will execute the original line unchanged.

Automatic correction is not guaranteed to work the way the user intended. Command line parsing is done in a rudimentary fashion. It is mostly provided as an experimental feature. Suggestions and improvements are welcome.

Documentation Lookup

The editor function `run-help` (`M-h`) prints a help file on the current command (using the same definition of current as the completion routines use). This help file is found by searching the path list `HPATH` for files of the form `foo.help`, `foo.1`, `foo.8`, or `foo.6` in that order (assuming that the current command is `foo`). The file is just printed, not paged in any way. This is because `run-help` is meant to be used to look up short help files, not manual pages (although it can do manual pages also).

Enhanced History Mechanism

`tsch` will save the history list between login sessions. It does this by writing the current list to the file `~/.history` on logout, and reading it in on login. For example, placing the line

```
> set history=25 savehist=20
```

tells `tcsh` to save the last 25 commands on the history list, and to save the last 20 of them between logins. The `savehist` variable may be set up to the size of history, although it is an error to have `savehist` larger than `history.` In addition to the above `tcsh,` keeps unparsed (literal) versions of the history if the variable `histlit` is set. Also the toggle-history function toggles between the parsed and literal version of the recalled history in the editor buffer. For example:

```
> set histlit
> echo !:s/foo/bar; ls
Modifier failed.
> ^P
> echo !:s/foo/bar; ls
> unset histlit
> echo !:s/foo/bar; ls
Modifier failed.
> ^P
> echo unset histlit[M-r]
> echo !:s/foo/bar; ls
```

`tsch` also supports the history escape `!#.` This undocumented `csh` escape holds the words of the current line. This is useful in renaming commands:

```
> mv foo bar!#:1
mv foo barfoo
```

Care should be taken when using this history expansion in `csh` since there is no check for recursion. In `tcsh` up to 10 levels of recursion are allowed.

Another difference between `tcsh` and `csh` history expansion, is the treatment of history arguments. In `csh !3d` expands to event 3 with the letter `d` appended to it. There is no way to repeat a command that begins with a number using the name of the command in the history escape. In `tcsh` only numeric arguments are treated as event numbers; therefore `!3d` is interpreted as: repeat the last command that started with the string `3d`. To mimic the `csh` behavior `!3\d` can be used.

Automatic Logout/Locking

The automatic locking and logout time are controlled by the variable `autologout`. The value of the first word is the number of minutes of inactivity will be allowed before automatically logging the user out. When that many minutes have been reached, the shell prints `auto-logout` and dies (without executing `~/.logout`). The default for `tcsh` is to set `autologout` for 60 minutes on login shells, and when the user is root. To disable autologout (for instance in a window system), unset the shell variable `autologout`.

The value of the optional second word indicates the number of minutes of inactivity after which the user will be prompted for his login password to continue working. Failure to enter the correct password 5 times, results in auto-logout.

Execution Of A Command Prior To Each Prompt

`tsch` supports a special alias, `precmd`, which if set holds a command that will be executed before printing each prompt. For example, if the user has done

```
> alias precmd date
```

then the program `date` will be run just before the shell prompts for each command. There are no limitations on what precmd can be set to do, although discretion should be used.

Periodic Command Execution

`tsch` is now capable of providing periodic command execution through the use of the shell variable `tperiod` and the alias `periodic`. When these items are set, the alias `periodic` will be executed every `tperiod` minutes. This provides a convenient means for checking on common but infrequent changes, such as new messages. Example:

```
> set tperiod = 30
> alias periodic checknews
```

This will cause the `checknews` program to be run every 30 minutes. Having the alias `periodic` set but with an unset `tperiod` (or a value of `0` for `tperiod)` will cause `periodic` to degenerate to another form of `precmd.`

New Prompt Format

The format for the `prompt` shell variable has been changed to include many new things, such as the current time of day, current working directory, etc. The new format uses `%<char>` to signal an expansion, much like `printf(3S)`.

The available sequences are presented in the table “New Prompt Format Sequences”.

Table 6: New Prompt Format Sequences

Prompt Format	Description
`%/`	Current working directory
`%~`	`cwd`. If it starts with `$HOME`, that part is replaced by a ~. In addition if a directory name prefix matches a user’s home directory, that part of the directory will be substituted with `~user`. NOTE: The `~user` substitution will only happen if the shell has performed a ~ expansion for that user name in this session
`%c` or `%.`	Trailing component of `cwd`, may be followed by a digit to get more than one component, if it starts with `$HOME`, that part is replaced with a ~
`%C`	Trailing component of `cwd`, may be followed by a digit to get more than one component, no ~ substitution
`%h`, `%!`, `!`	Current history event number
`%M`	The full machine hostname
`%m`	The hostname up to the first
`%S (%s)`	Start (stop) standout mode
`%B (%b)`	Start (stop) boldfacing mode. (Only if `tcsh` was compiled to be eight bit clean.)
`%U (%u)`	Start (stop) underline mode. (Only if `tcsh` was compiled to be eight bit clean.)
`%t` or `%@`	Current time of day, in 12-hour, am/pm format.
`%T`	Current time of day, in 24-hour format. (But see the `ampm` shell variable below.)
`\c`	c is parsed the same way as in bindkey.
`^c`	c is parsed the same way as in bindkey.
`%%`	A single `%`.
`%n`	The user name, contents of `$user`.

Table 6: New Prompt Format Sequences

Prompt Format	Description
`%d`	The weekday in `<Day>` format.
`%D`	The day in `dd` format.
`%w`	The month in `<Mon>` format.
`%W`	The month in `mm` format.
`%y`	The year in `yy` format.
`%Y`	The year in `yyyy` format.
`%l`	The line (`tty`) the user is logged on.
`%L`	clear from prompt to end of display or end of line.
`%#`	A # if `tcsh` is run as a root shell, a > if not.
`%{..%}`	Include string as a literal escape sequence. Note that the enclosed escape sequence, should only be used to change terminal attributes and should not move the cursor location. Also, this cannot be the last character in the prompt string. (Available only if `tcsh` was compiled to be eight bit clean.)
`%?`	return code of the last command executed just before the prompt.
`%R`	In prompt3 this is the corrected string; in `prompt2` it is the status of the parser.

The sequences for standout are often used to indicate that this is an enabled (running as root) shell. An example:

```
.ie "t" "\{>" "set" "prompt="%m" "[%h]" "%B[%@]%b" "[%/]" "you"
"rang?"
tut [37] [2:54pm] [/usr/accts/sys] you rang? _

.\}
.el "\{>" "set" "prompt="%m" "[%h]" "%U[%@]%u" "[%/]" "you"
"rang?" """
tut [37] [2:54pm] [/usr/accts/sys] you rang? _

.\}
```

In addition, there is a new variable, `prompt2`, which is used to prompt for the body of `while` and `for` loops (wherever normal `csh` prompts with a question mark). The default for `prompt2` is `%R?`: the status of the parser followed by a question mark. This alternate prompt is also used when the parser is waiting for more input; i.e. when the previous line ended in a `\`. The `prompt3` variable is used when displaying the corrected command line when automatic spelling correction is in effect; it defaults to `CORRECT>%R (y|n|e)?`.

Time-Stamped History List

The history list in `tcsh` now has a time-of-day stamp attached to each history list event. This time stamp is printed whenever the history command is executed. This allows the user to keep track of when the various events occurred. The time stamps are not maintained on the saved history list (also available in `csh`); thus, on logging back in, all the saved history events will be recorded with the login time as their time stamp. The time stamp printouts can be omitted from the history list by adding the -t switch to the `history` command.

Directory Access

`tsch` supports three new flags to control directory style printing for `cd`, `pushd`, `popd`, and `dirs`:

`-n`	Print entries in new lines so that the screen width is not exceeded
`-l`	Don't print ~ but print the whole path
`-v`	Print the stack entries one in each line, preceded by the stack number.

Note that `popd +n` can be used to pop out stack entries of directories that do not exist any more.

`tsch` always saves the last directory visited. Changing to the previous directory can be done by using

```
'cd "-'"
```

or

```
'pushd "-'."
```

`tsch` will now allow the user to access all elements in the directory stack directly. The syntax `=<digit>` is recognized by `tcsh` as indicating a particular directory in the stack. (This works for the file/command name recognition as well.) This syntax is analogous to the ~ syntax for access to users' home directories. The stack is viewed as zero-based, i.e., `=0` is the same as `$cwd`, which is the same as `.` (period).

As a special case, the string =– is recognized as indicating the last directory in the stack. Thus,

```
> dirs -v
0 /usr/net/bin
1 /usr/spool/uucp
2 /usr/accts/sys
> echo =2
/usr/accts/sys
> ls -l =1/LOGFILE
-rw-r--r-- 1 uucp 2594 Jan 19 09:09 /usr/spool/uucp/LOGFILE
> echo =-/.cs*
/usr/accts/sys/.cshrc
> echo =4
Not that many dir stack entries.
>
```

`tsch` will complain if you ask for a directory stack item which does not exist.

In the normal `csh`, saying `pushd +2` would rotate the entire stack around through 2 stack elements, placing the entry found there at the top of the stack. If, however, the new shell variable `dextract` is set, then issuing `pushd +n` will cause the nth directory stack element to be extracted from its current position, which will then be pushed onto the top of the stack. Example:

```
> dirs
~ /usr/spool/uucp /usr/net/bin /sys/src
> set dextract
> pushd +2
/usr/net/bin ~ /usr/spool/uucp /sys/src
```

```
> unset dextract
> pushd +2
/usr/spool/uucp /sys/src /usr/net/bin ~
```

In addition, if the shell variable `dunique` is set, only directories that are not already in the directory stack will be added when `push` is invoked.

The way symbolic links that point to directories are crossed is determined by the `symlinks` variable. By default, it is not set to anything and symbolic links are not treated specially. However, it can be set to either: `chase`, `ignore`, or `expand`. If `symlinks` is set to `chase`, then every time the directory changes, `$cwd` reflects the real directory name, and not the name through the link.

A notable exception is the user's home directory, but that should be fixed. If `symlinks` is set to either `ignore` or `expand`, then `tcsh` tries to determine the current directory name relative to the name of the current directory before the link was crossed. In simple terms, if you `chdir` through a symbolic link and then `cd ..`, you will end `..` relative to where you were before you crossed the link and not `..` relatively to where the symbolic link points. While `ignore` affects only builtin commands and completion, `expand` affects command execution too, by fixing arguments that it considers to be path names.

Unfortunately, this does not work in every case, especially when file arguments are embedded in command line options. `Expand` also allows to specify the real (as opposed to the one determined relatively to `$cwd`), by quoting the file argument.

For example:

```
> cd /tmp
> mkdir from from/src to
> ln -s ../from/src to/dst
> unset symlinks
> cd /tmp/to/dst; echo $cwd
/tmp/to/dst
> cd ..; echo $cwd
/tmp/from
> set symlinks=chase
> cd /tmp/to/dst; echo $cwd
/tmp/from/src
> cd ..; echo $cwd
/tmp/from
> set symlinks=ignore
> cd /tmp/to/dst; echo $cwd
/tmp/to/dst
> cd ..; echo $cwd
/tmp/to
> set symlinks=expand
> cd /tmp/to/dst; echo $cwd
/tmp/to/dst
> cd ..; echo $cwd  # builtin
/tmp/to
> cd /tmp/to/dst; echo $cwd
/tmp/to/dst
> cd ".."; echo $cwd  # quoted
/tmp/from
> /bin/echo ..   # rewrites non builtin commands.
/tmp/to
> /bin/echo ".."
```

While the `expand` setting is the most convenient, and makes symbolic links invisible, it can sometimes be misleading and confusing in cases where it fails to recognize arguments. A compromise is to use the `ignore` setting and use the editor functions normalize-path, bound by default to `^Xn`.

Watching For Logins And Logouts

`tsch` has a mechanism so that the user can watch for login and logout activity of any user or terminal in the system. This is accomplished using the new special shell variable `watch`, which contains login/terminal pattern pairs to be checked for activity. For example:

```
> set watch=(sys ttyjd root console)
```

This setting will allow the user to check on when the user `sys` logs in on `/dev/ttyjd`. Similarly, it will inform the user of root's activity on the console. In order to be more general, the word `any` may be substituted for either a user's or a terminal's name, thus allowing

```
> set watch=('b*' any any ttyh0)
```

which will check for users that match the pattern `b*` logging in or out of the system on any terminal, as well as anyone logging in to `/dev/ttyh0`. Naturally, the completely general case

```
> set watch=(any any)
```

allows the user to check on any and all login/logout activity in the system.

By default, the interval between checks of users on the system is 10 minutes; this can be changed by making the first element of `watch` a number of minutes which should be used instead, as in

```
> set watch=(40 any any)
```

which will check for any users logging in or out every 40 minutes.

There is also a new command, `log`, which is used to cause `tcsh` to inform the user of all users/terminals affected by `watch` whether they have been announced before or not. This is useful if a user has been on for some time and cannot remember if a particular person/terminal is online right now or not. `Log` will reset all indication of previous announcement and give the user the login list all over again, as well as printing the current value of `watch`.

The first time that watch is set at tcsh startup, all affected users and terminals will be printed as though those users/terminals had just logged on. This may appear to be a bug, but is generally considered a feature, since it allows the user, when first logging in, to see who is on the system.

The format of the printouts can be tailored via setting of the variable who. The sequences in the table “Sequences For Format Specification” are available for the format specification.

Table 7: Sequences For Format Specification

Sequence	Description
%n	The name of the user that logged in/out
%a	The observed action, i.e. logged on, logged off, or replaced <olduser> on
%l	The line (tty) the user is logged on
%S (%s)	Start (stop) standout mode
%B (%b)	Start (stop) boldfacing mode. (Only if tcsh was compiled to be eight bit clean)
%U (%u)	Start (stop) underline mode. (Only if tcsh was compiled to be eight bit clean)
%M	The full hostname of the remote host (local if non-remote)
%m	The hostname up to the first .. If only the ip address is available or the utmp field contains the name of an x-windows display, the whole name is printed
%t or %@	The time, in 12-hour, am/pm format (logout time is approximated if unavailable)
%T	The time, in 24-hour format. (but see the ampm shell variable below)
%D	The day in dd format
%w	The month in <Mon> format
%W	The month in mm format
%Y	The year in yy format

The `%M` and `%m` sequences are only available on systems that store the remote hostname in `/etc/utmp`. If `who` is not set, the format defaults to `%n has %a %l from %m.`, or `%n has %a %l.` on systems that don't store the hostname.

Timed Event List

`tsch` now supports a scheduled-event list through the use of the command `sched`. This command gives the user a mechanism by which to arrange for other commands to be executed at given times. An event is added to the scheduled-event list by saying

```
> sched [+]hh:mm <command>
```

as in

```
> sched 11:00 echo It\'s eleven o\'clock.
```

This will make an entry in the list at 11am for the echo command to be run with the given arguments. The time may be specified in either absolute or relative time, and absolute times may have a morning/afternoon specification as well, using `am` or `pm`. For example,

```
> sched +2:15 /usr/lib/uucp/uucico -r1 -sother
> sched 5pm set prompt='[%h] It\'s after 5; go home: >'
> sched +3am echo This syntax doesn\'t work.
Relative time inconsistent with am/pm.
>
```

Note that `tcsh` will complain if you try to make faulty time specifications.

Printing the current time-event list is accomplished by giving the `sched` command with no arguments:

```
> sched
  1 Wed Apr 4 15:42 /usr/lib/uucp/uucico -r1 -sother
  2 Wed Apr 4 17:00 set prompt=[%h] It's after 5;
    go home: >
>
```

There is also a mechanism by which the user can remove an item from the list:

```
> sched --3
Usage for delete: sched -<item#>.
> sched -3
Not that many scheduled events.
> sched -2
> sched
  1 Wed Apr 4 15:42 /usr/lib/uucp/uucico -r1 -sother
>
```

All commands specified on the scheduled-event list will be executed just prior to printing the first prompt immediately following the time when the command is to be run. Hence, it is possible to miss the exact time when the command is to be run, but `tcsh` will definitely get around to all commands which are overdue at its next prompt. Scheduled-event list items which come due while `tcsh` is waiting for user input will be executed immediately. In no case, however, will normal operation of already-running commands be interrupted so that a scheduled-event list element may be run.

This mechanism is similar to, but not the same as, the `at` command on some UNIX systems. Its major disadvantage is that it does not necessarily run a command at exactly the specified time (but only if another command is already being run). Its major advantage is that commands which run directly from `tcsh`, as `sched` commands are, have access to shell variables and other structures. This provides a mechanism for changing one's working environment based on the time of day.

Builtin For ls -F

There is a new builtin command called `ls-F` which does the same thing as the command `ls -F`. If the shell variable `showdots` has been set, it acts like `ls -aF`. If `showdots` is equal to `-A`, then `ls-F` acts like `ls -AF`. `ls-F` works like `ls`, only it is generally faster, since it does not need to fork and exec, and safer against Trojan horses if the user has `.` in the execution path. If other switches are passed to `ls-F`, then the normal ls is executed. Aliasing `ls` to `ls-F` provides a fast alternative way of listing files. Note that on non-BSD machines, where `ls -C` is not the default, `ls-F`, behaves like `ls -CF`. `ls-F` appends the characters depending on the file type, as shown in the table "ls-F Appended Characters".

Table 8: ls-F Appended Characters

<table>
<tr><th>Character</th><th>Description</th></tr>
<tr><td>=</td><td>File is an AF_UNIX domain socket. [if system supports sockets]</td></tr>
<tr><td>|</td><td>File is a named pipe (fifo) [if system supports named pipes]</td></tr>
<tr><td>%</td><td>File is a character device</td></tr>
<tr><td>#</td><td>File is a block device</td></tr>
<tr><td>/</td><td>File is a directory</td></tr>
<tr><td>*</td><td>File is executable</td></tr>
<tr><td>+</td><td>File is a hidden directory [aix]or context dependent [hpux]</td></tr>
<tr><td>:</td><td>File is network special [hpux]</td></tr>
</table>

On systems that support symbolic links the variable `listlinks` controls the way symbolic links are identified. If `listlinks` is not set then the character `@` is appended to the file. If `listlinks` is set, then a character is appended to the filename depending on the type of file the symbolic link references, as shown in the table "listlink Append Characters".

Table 9: `listlink` Append Characters

Character	Description
@	File is a symbolic link pointing to a non-directory
>	File is a symbolic link pointing to a directory
&	File is a symbolic link pointing to nowhere

While setting `listlinks` can be helpful while navigating around the file system, it slows down `ls-F` and it causes mounting of file systems if the symbolic links point to an NFS auto-mounted partition.

Globbing Syntax Additions

The syntax for any character in a range (for example `.[a-z]*`) has been extended so as to conform with standard UNIX regular expression syntax (see `ed`). Specifically, after an open bracket (`[`), if the first character is a caret (`^`), then the character matched will be any not in the range specified. For example:

```
> cd ~
> echo .[a-z]*
.cshrc .emacs .login .logout .menuwmrc
> echo .[^.]*
.Xdefaults .Xinit .cshrc .emacs .login .logout .menuwmrc
>
```

Note that the second form includes .Xdefaults and .Xinit because 'X' (and all the lower case letters) are outside of the range of a single `.`(period).

Also the ability to negate a globbing pattern has been added:

```
> echo *
foo foobar bar barfoo
> echo ^foo*
bar barfoo
```

Note that this does not work correctly if the expression does not have any wildcard characters (`?*[]`) or if the expression has braces `{}`.

New Environment And Shell Variables

On startup, `tcsh` now automatically initializes the environment variable `HOST` to the name of the machine that it is running on. It does this by doing a `gethostname()` system call, and setting `HOST` to the result.

`tsch` also initializes the environment variable `HOSTTYPE` to a symbolic name for the type of computer that it is running on. This is useful when sharing a single physical directory between several types of machines (running NFS, for instance). For example, if the following is in `.login`:

```
set path = (~/bin.$HOSTTYPE /usr/ucb /bin /usr/bin /usr/games .)
```

and the user has directories named `bin.`*machine* (where *machine* is a name from the list below), then the user can have the same programs compiled for different machines in the appropriate `bin.`*machine* directories and `tcsh` will run the binary for the correct machine.

The possible values are shown in the table “/bin Directories For Various Platforms”.

Table 10: `/bin` Directories For Various Platforms

machine Name	Platform	machine Name	Platform
aix370	an IBM 370, running aix	mac2	an Apple Computer Macintosh II, running AUX
aixESA	an IBM ESA, running aix	mascomp	a Concurrent (Masscomp), running RTU
alliant	an Alliant FX series	minix386	an i386 running minix
amdahl	an Amdahl running uts 2.1	minix	a mac or an amiga running minix
amiga	an amiga running amix 2.02	mips	another mips CPU
apollo	an Apollo running DomainOS	multimax	an Encore Computer Corp. Multimax (32000 based)
att3b15	an AT&T 3b15	news	a Sony NEWS 800 or 1700 workstation
att3b2	an AT&T 3b2	news_mips	a NeWS machine with mips CPU
att3b20	an AT&T 3b20	nd500	a Norsk Data ND 500/5000 running Ndix
att3b5	an AT&T 3b5	ns32000	an NS32000 CPU machine
balance	a Sequent Balance (32000 based)	next	a NeXT computer
butterfly	a BBN Computer Butterfly 1000	pfa50	a PFU/Fujitsu A-xx computer
convex	a Convex	ps2	an IBM PS/2, running aix

Table 10: `/bin` Directories For Various Platforms

machine Name	Platform	machine Name	Platform
cray	a Cray running Unicos	ptx	a Sequent Symmetry running DYNIX/ptx (386/486 based)
decstation	a DecStation XXXX	pyramid	a Pyramid Technology computer (of any flavor)
eta10	an ETA 10 running SVR3	rs6000	an IBM RS6000, running aix
gould-np1	a Gould NP1	rt	an IBM PC/RT, running BSD (AOS 4.3) or mach
hk68	a Heurikon HK68 running Uniplus+ 5.0	rtpc	an IBM PC/RT, running aix
hp300	an HP 9000, series 300, running mtXinu	sco386	an Intel 386, running SCO
hp800	an HP 9000, series 800, running mtXinu	stellar	a stellar running stellix
hp9000s300	an HP 9000, series 300, running hpux	sun	a Sun workstation of none of the above types
hp9000s500	an HP 9000, series 500, running hpux	sun2	a Sun Microsystems series 2 workstation (68010 based)
hp9000s700	an HP 9000, series 700, running hpux	sun3	a Sun Microsystems series 3 workstation (68020 based)
hp9000s800	an HP 9000, series 800, running hpux	sun386i	a Sun Microsystems 386i workstation (386 based)
hp	an HP, running hpux	sun4	a Sun Microsystems series 4 workstation (SPARC based)
i386	an Intel 386, generic	symmetry	a Sequent Symmetry running DYNIX 3 (386/486 based)
i386-mach	an Intel 386, running mach	tahoe	a tahoe running 4BSD

Table 10: /bin Directories For Various Platforms

machine Name	Platform	machine Name	Platform
intel386	an Intel 386, running INTEL's SVR3	titan	an Stardent Titan
iris3d	a Silicon Graphics Iris 3000	unixpc	an UNIX/PC running SVR1 att7300 aka att3b1
iris4d	a Silicon Graphics Iris 4D	vax	a Digital Equipment Corp. Vax (of any flavor)
isc386	an Intel 386, running ISC	vistra800	a Stardent Vistra 800 running SVR4
m88k	an mc88000 CPU machine		

(The names of the machines are usually trademarks of the corresponding companies.)

`tsch` also initializes the shell variables `uid` and `gid` to the value of the current real user ID/GID. This is useful for telling what user/group the shell is running as. Under Domain/OS `tcsh` will also set `oid` indicating the current real organization id.

Commands For Debugging

Only two such commands are available at this point, both concerned with testing `termcap` entries.

`telltc` tells you, politely, what `tcsh` thinks of your terminal, and `settc cap value` tells `tcsh` to believe that the termcap capability `cap` (as defined in `termcap`) has the value `value`. No checking for sanity is performed, so beware of improper use.

Note that Concept terminal users may have to `settc xn no` to get proper wrapping at the right-most column.

Searching For The Visual History

Four new editor functions have been added in emacs mode: Two of them, history-search-backward, bound to `M-p` (and `M-P`), and history-search-forward, bound to `M-n` (and `M-N`), implement a conventional search mechanism, while the other two, `i-search-back` and `i-search-fwd`, both unbound by default, implement the emacs-style incremental search.

History-search-backward (history-search-forward) searches backward (forward) through the history list for previous (next) occurrence of the beginning of the input buffer as a command. That is, if the user types:

```
> echo foo
foo
> ls
filea  fileb
> echo bar
bar
>
```

and then types `echo<ESC>p`, the shell will place `echo bar` in the editing buffer. If another `M-p` was entered, the editing buffer would change to `echo foo`. This capability is compatible with the plain visual history; if the user were to then enter `^P` the editing buffer would be changed to `ls`. The pattern used to search through the history is defined by the characters from the beginning of the line up to the current cursor position and may contain a shell globbing pattern. Successive history searches use the same pattern.

When the incremental search functions are used, a prompt - `bck:` or `fwd:`, depending on the chosen direction - appears, and characters typed are appended to the search pattern. The matching command line is placed in the editing buffer, with the cursor positioned at the point where the pattern was found (the incremental search will also do searching within a command line). If the search fails, the terminal bell is rung, and the `:` in the prompt is changed to `?`.

If these functions are bound to single characters, and one of those is typed again during the incremental search, the search continues in the corresponding direction, with the pattern unchanged; if the previous search failed, this will also cause it to wrap around the history list. Finally, the special meaning of special characters is shown in the table "Special Meaning Of Search Characters".

Table 11: Special Meaning Of Search Characters

Search Characters	Description
`Delete`	(or any character bound to `backward-delete-char`) Undo the effect of the last character typed, deleting a character from the pattern if appropriate
`^G`	Abort the search, back to the last successful one if previous search failed, otherwise entire search
`^W`	Append the rest of the current word to the pattern
`ESC`	Terminate the search, leaving current line in buffer

Any other character, that isn't bound to `self-insert-command`, terminates the search, leaving the current command line in the editing buffer, after which `tcsh` interprets the character as if it was normal input - notably `Return` will cause the current command line to be executed.

In vi mode `vi-search-fwd` (bound to `/`) and `vi-search-back` (bound to `?`) can be used to search the visual history. In command mode `/` or `?` followed by a shell pattern, searches the visual history for the pattern. The pattern can be terminated either with a `Return`, in which case the matched history event is placed in the command buffer or an `Escape`, in which case the matched history event is executed. If no history event matches the shell pattern then the bell is rung.

Builtin Which/Where Commands

There is now a builtin version of the `which` command. The builtin version is just like the original, except that it correctly reports aliases peculiar to this `tcsh`, and builtin commands. The only other difference is that the builtin runs somewhere between 10 and 100 times faster. There is also a key-function interface to this command: the `which-command` function (normally bound to `M-?`), can be used anywhere on the command line, and will in effect do a `which` on the command name.

Similarly where will report all known instances of the command: aliases, builtins, as well as all references in the path.

Restarting A Stopped Editor

There is another new editor function: `run-fg-editor`, which is bound to `M-^Z`. When typed, it saves away the current input buffer, and looks for a stopped job with a name equal to the file name part (last element) of either the `EDITOR` or `VISUAL` environment variables (if not defined, the default names are `ed` and `vi` respectively). If such a job is found, then it is restarted as if `fg %name` had been typed. This is used to toggle back and forth between an editor and the shell easily. Some people bind this function to `^Z` so they can do this even more easily.

Status Line Support

`tsch` has a new builtin called `echotc` that allows the user to access the terminal capabilities from the command line, similar to the system V `tput`.

```
> echotc home
```

Places the cursor at the home position and

```
> echotc cm 3 10
```

places the cursor at column 3 row 10. This command replaces the `el` and `sl` variables that used contain the escape sequences to begin and end status line changes. The command:

```
> echo $sl this is a test $el
```

is replaced by:

```
> echotc ts 0; echo "this is a test"; echotc fs
```

In addition `echotc` understands the arguments `baud, lines, cols, meta,` and `tabs` And prints the baud rate, the number of lines and columns, and `yes` or `no` depending if the terminal has tabs or a meta key.

This can be useful in determining how terse the output of commands will be depending on the baud rate, or setting limits to commands like history to the highest number so that the terminal does not scroll:

```
> set history=`echotc lines`
> @ history--
```

Note: Termcap strings may contain wildcard characters, and echoing them will not work correctly. The suggested method of setting shell variables to terminal capability strings is using double quotes, as in the following example that places the date in the status line:

```
> set tosl="`echotc ts 0`"
> set frsl="`echotc fs`"
> echo -n "$tosl";date; echo -n "$frsl"
```

`Echotc` accepts two flags. The flag `-v` enables verbose messages and the flag `-s` ignores any errors and returns the empty string if the capability is not found.

Execution Of A Command After Changing The Current Working Directory

`tsch` now supports a special alias, `cwdcmd`, which if set holds a command that will be executed after changing the value of $cwd. For example, if the user is running on an X window system `xterm`, and a re-parenting window manager that supports title bars such as `twm and has done`:

```
> alias cwdcmd 'echo -n "^[]2;${HOST}:$cwd ^G"'
```

then the shell will change the title of the running `xterm` to be the name of the host, a colon, and the full current working directory. A more fancy way to do that is:

```
> alias cwdcmd 'echo -n "^[]2;${HOST}:$cwd^G^[]1;${HOST}^G"'
```

This will put the hostname and working directory on the title bar but only the hostname in the icon manager menu. Note that if a user defines `cwdcmd` to contain a `cd`, `pushd`, or `popd`, command, an infinite loop may result. In this case, it is the author's opinion that said user will get what he deserves.

Native Language System

`tsch` is eight bit clean (if so compiled, see the description of the `version` shell variable below), and will thus support character sets needing this capability. The `tcsh` support for NLS differs depending on whether it was compiled to use the system's NLS (again, see the `version` variable) or not. In either case, the default for character classification (i.e. which characters are printable etc.) and sorting is 7-bit ASCII, and any setting or unsetting of the `LANG` or `LC_CTYPE` environment variables will cause a check for possible changes in these respects.

When using the system's NLS, the `setlocale` C library function will be called to determine appropriate character classification and sorting - this function will typically examine the `LANG` and `LC_CTYPE` variables for this purpose (refer to the system documentation for further details). Otherwise, NLS will be simulated, by assuming that the ISO 8859-1 character set is used whenever either of the `LANG` and `LC_CTYPE` variables are set, regardless of their values. Sorting is not affected for the simulated NLS.

In addition, with both real and simulated NLS, all printable characters in the range \200-\377, i.e. those that have M-<char> bindings, are automatically rebound to `self-insert-command` (the corresponding binding for the `escape+<char>` sequence, if any, is left alone). This automatic rebinding is inhibited if the `NOREBIND` environment variable is set - this may be useful for the simulated NLS, or a primitive real NLS which assumes full ISO 8859-1 (otherwise all M-<char> bindings in the range \240-\377 will effectively be undone in these cases). Explicitly rebinding the relevant keys, using `bindkey`, is of course still possible.

Unknown characters (i.e. those that are neither printable nor control characters) will be printed using the `\nnn` format. If the `tty` is not in 8 bit mode, other 8 bit characters will be printed by converting them to ASCII and using standout mode. `tsch` will never change the 7/8 bit mode of the `tty`, and will track user-initiated settings for this - i.e. it may be necessary for NLS users (or, for that matter, those that want to use a Meta key) to explicitly set the `tty` in 8 bit mode through the appropriate `stty` command in e.g. the `.login` file.

Automatic Process Time Reporting

Automatic process time reporting is a feature that exists in `csh`, but it is usually not documented. In addition `tcsh` provides a slightly enriched syntax. Process time reports are controlled via the `time` shell variable. The first word of the `time` variable indicates the minimum number of CPU seconds the process has to consume before a time report is produced. The optional second word controls the format of the report. The sequences from the table "Sequences For Format Specification" are available for the format specification.

Table 12: Sequences For Format Specification

Sequence	Description
%U	The time the process spent in user mode in cpu seconds.
%S	The time the process spent in kernel mode in cpu seconds.
%E	The elapsed time in seconds.
%P	The CPU percentage computed as (`%U + %S`) `/ %E`.

The sequences from the table "Sequences For Systems With BSD Resource Limit Functions" are supported only in systems that have the BSD resource limit functions.

Table 13: Sequences For Systems With BSD Resource Limit Functions

Sequence	Description
%W	Number of times the process was swapped
%X	The average amount in (shared) text space used in KB
%D	The average amount in (unshared) data/stack space used in KB
%K	The total space used (`%X + %D`) in KB
%M	The maximum memory the process had in use at any time in KB
%F	The number of major page faults (page needed to be brought from disk)
%R	The number of minor page faults
%I	The number of input operations
%O	The number of output operations

Table 13: Sequences For Systems With BSD Resource Limit Functions

Sequence	Description
%r	The number of socket messages received
%s	The number of socket messages sent
%k	The number of signals received
%w	Number of voluntary context switches (waits)
%c	Number of involuntary context switches

The default time format is `%Uu %Ss %E %P %X+%Dk %I+%Oio %Fpf+%Ww` for systems that support resource usage reporting and `%Uu %Ss %E %P` for systems that do not.

For Sequent's DYNIX/ptx `%X, %D, %K, %r` and `%s` are not supported. However, the additional sequences from the table "Sequent's DYNX/ptx Sequences" are available.

Table 14: Sequent's DYNX/ptx Sequences

Sequences	Decription
%Y	The number of system calls performed.
%Z	The number of pages which are zero-filled on demand
%i	The number of times a process' resident set size was increased by the kernel
%d	The number of times a process' resident set size was decreased by the kernel
%l	The number of read system calls performed.
%m	The number of write system calls performed.
%p	The number of reads from raw disk devices.
%q	The number of writes to raw disk devices.

The default time format for Sequent's DYNIX/ptx is `%Uu %Ss $E %P %I+%Oio %Fpf+%Ww`. Also note that the CPU percentage can be higher than 100% on multi-processors.

OS/Dependent Builtin Support

TRANSPARENT COMPUTING FACILITY. On systems that support TCF (aix-ibm370, aix-ps2) the builtins from the table “Added Builtin Support” have been added.

Table 15: Added Builtin Support

<table>
<tr><th>Builtin</th><th>Description</th></tr>
<tr><td><code>getspath</code></td><td>Print the current system execution path</td></tr>
<tr><td><code>setspath LOCAL|<site>|<cpu> ...</code></td><td>Set the current execution path</td></tr>
<tr><td><code>getxvers</code></td><td>Print the current experimental version prefix</td></tr>
<tr><td><code>setxvers [<string>]</code></td><td>If the optional string is omitted, any experimental version prefix is removed. Otherwise the experimental version prefix is set to string</td></tr>
<tr><td><code>migrate [-<site>] <pid>| %<jobid> ... migrate -<site></code></td><td>The first form migrates the process or job to the site specified or the default site determined by the system path. The second form, is equivalent to migrate -<site> $$, i.e. migrates the current process to the site specified. Note: migrating <code>tcsh</code> itself can cause unexpected behavior, since the shell does not like to lose its tty</td></tr>
</table>

In addition, jobs will print the site the job is executing.

Domain/OS Support	
`inlib <shared-library> ...`	`inlib` adds shared libraries to the current environment. There is no way to remove them...
`rootnode //<nodename>`	Change the name of the current rootnode. From now on, `/` will resolve to `//<rootnode>`

`ver [<systype>] [<command>]`	Without arguments, print `$SYSTYPE`; with the `<systype>` provided, set `SYSTYPE` to the one provided. Valid systypes are `bsd4.3` and `sys5.3`. If a `<command>` is argument is given, then `<command>` is executed under the `<systype>` specified
Mach	
`setpath <path-spec> ...`	XXX: What does it do?
Masscomp/RTU	
`universe <universe-spec> ...`	Sets the current universe to the specified parameter
Convex/OS	
`warp [<universe-spec>] ...`	Without arguments prints the current value of the universe. With a universe argument it sets the current universe to the value of the argument

Window Size Tracking

On systems that support `SIGWINCH` or `SIGWINDOW`, `tcsh` adapts to window resizing automatically and adjusts the environment variables `LINES` and `COLUMNS` if set. Also if the environment variable `TERMCAP` contains `li#`, and `co#` fields, these will be adjusted also to reflect the new window size.

Input Files

On startup `tcsh` will try to source `/etc/csh.cshrc` and then `/etc/csh.login` if the shell is a login shell. Then it will try to source `$HOME/.tcshrc` and then `$HOME/.cshrc` if `$HOME/.tcshrc` is not found. Then it will source `$HOME/.login` if the shell is a login shell. On exit `tcsh` will source first `/etc/csh.logout` and then `$HOME/.logout` if the shell was a login shell.

Note: On convexos, stellix, and INTEL the names of the system default files are `/etc/cshrc`, `/etc/login`, and `/etc/logout`, respectively. On NeXT, the names are also different: `/etc/cshrc.std`, `/etc/login.std`, and `/etc/logout.std`. Finally, on irix, A/UX, AMIX, and the cray only the file `/etc/cshrc` is executed if the shell is a login shell.

Command Line Options

This section describes options that are either undocumented in `csh (*)` or present only in `tcsh. (+)`

`-d`	Load `~/.cshdirs` (If `tcsh` was compiled with `CSHDIRS` enabled)(+)
`-Dname[=value]`	Set environment variable name to value. (Domain/OS only) (+)
`-F`	Use `fork()` instead of `vfork()` to spawn processes. (Convex/OS only) (+)
`-l`	Make `tcsh` behave like a login shell. (+)
`-m`	Allow reading of a .cshrc that does not belong to the effective user. Newer versions of `su` can pass that to the shell. (some versions of `csh` have it) (+*)
`-q`	Make the shell accept `SIGQUIT`, and behave when it is used under a debugger. Job control is disabled. (*)

History And Variable Modifier Enhancements

`tsch` accepts more than one variable modifier per variable or history expansion. For example, in `csh` the following command expands to:

```
% set a=/usr/local/foo.bar.baz
% echo $a:t:r:e
foo.bar.baz:r:e
```

but in `tcsh:`

```
> set a=/usr/local/foo.bar.baz
> echo $a:t:r:e
bar
```

This bug fix changes slightly the input syntax of `csh`, causing expressions of the form to have invalid syntax:

```
> set a=/usr/local/foo.bar.baz
> echo $a:t:$cwd
Unknown variable modifier.
```

Which is the correct behavior, since after the second colon a variable modifier is expected and '$' is found. Expressions like this should be rewritten as:

```
> echo ${a:t}:$cwd
```

`tsch` can also use the `l` and `u` modifiers to change the case of a word.

Also `tcsh` has an additional : character `a` which applies the current modifier multiple times in the same word, in the same way `g` applies the modifier once in each word:

```
> set foo=(a:b:c d:e:f)
> echo $foo:s/:/ /
a b:c d:e:f
> echo $foo:gs/:/ /
a b:c d e:f
> echo $foo:as/:/ /
a b c d:e:f
> echo $foo:ags/:/ /
a b c d e f
```

The `csh` expert will notice in the above example another `tcsh` enhancement. In `csh` the `s` modifier does not work for variables. The `a` character works correctly with other modifiers, for example:

```
> echo $host
tesla.ee.cornell.edu
> echo $host:r
tesla.ee.cornell
> echo $host:ar
tesla
```

Note: The a character causes a modifier to be applied to a word repeatedly until the modifier fails. This can lead to an infinite loop in the current implementation when used with the `s` modifier. For example `:as/f/ff/` will never terminate. This behavior might change in the future.

Programmable Completion

The new `complete` and `uncomplete` builtins can change the behavior of the listing and file-completion in `tcsh`. For example some commands like `cd`, `pushd`, `popd`, `mkdir`, `rmdir` accept only directories as arguments. Normally `tcsh` when asked to list or complete an argument for those commands takes into account all the files and not only directories. This behavior can be changed by telling `tcsh` that these commands only expect directory arguments:

```
> complete cd 'n/*/d/'
```

The above command tells `tcsh` to complete any following word with a directory.

The syntax for the `complete` and `uncomplete` builtins is:

```
complete [<command-pattern> [<completion-argument>]]
uncomplete <command-pattern>
```

Completion arguments are matched in turn from left to right and the first successful match is returned. There are three different completion arguments, modeled after the `sed` substitute command and they are of the form:

```
<command>/<match-pattern>/<complete-action>/[<suffix>|/]
```

The five commands available are:

`p` This specifies position-dependent completion. The matching pattern is a numeric range similar to the one used to index shell variables. If the current word is within that range, then the completion action is taken. For example:

```
> complete cd 'p/1/d/'
```

completes only the first argument of cd with a directory. A special useful case of position-dependent completion can be used to disambiguate command completion:

```
> set autolist
> co[TAB]
complete compress
> complete -co* 'p/0/(complete)/'
> co[TAB]
> complete _
```

The above command tells the completion mechanism to match any word on position 0, that matches the pattern co*. If a match is found, the word is replaced with the literal pattern complete. The leading – in the pattern signifies that this completion is to be used only with ambiguous commands.

n,N — This command specifies next-word completions. The matching pattern in this case is a shell pattern that must match the previous word on the command line (n), or the word before that (N). For example:

```
> complete find 'n/-user/u/'
```

Here the argument of the find command which follows the -user argument is completed from the list of users.

c,C — The last two completion commands complete the current word. The matching pattern in this case must match the prefix of the current word. For example:

```
> complete cc 'c/-I/d/'
```

completes the the word containing -I from the list of directories. While c discards the matched prefix, C keeps it and applies the completion using it.

So far the `d`, `u`, `(word)` completion actions have been mentioned. The complete list of completion actions are presented in the table “Full Listing Of Completion Actions”.

Table 16: Full Listing Of Completion Actions

Completion Action	Description	Completion Action	Description
`C`	Complete completions	`l`	Complete limits
`S`	Complete signals	`n`	Complete nothing
`a`	Complete aliases	`s`	Complete shell variable
`b`	Complete binding	`v`	Complete any variable
`c`	Complete command	`u`	Complete username
`d`	Complete directory	`x`	Explain what argument when listing is requested
`e`	Complete environment variable	`$var`	Complete using words from variable
`f`	Complete file or path component	`(...)`	Complete using words from list
`j`	Complete jobs	`` `...` ``	Complete evaluating word list from the output of command.

All the above completion actions, except `x`, `$var`, `(...)`, and `` `...` `` can have an additional shell pattern which must match the files to be completed, separated from the action with a `:`. The `x` completion action instead expects the explanation string specified this way. For example, in the C compiler, it makes sense to complete only files that the compiler understands:

```
> complete cc 'n/*/f:*.[cao]/'
```

It is also useful to exclude files from the completion list. Normally, `tcsh` will expand the following command:

```
> set fignore=(.o)
> ls
foo.c foo.o
```

```
> vi f[TAB]
> vi foo.c
> rm f[TAB]
> rm foo.c
```

A better way to do this is to use the complete builtin to protect precious files:

```
> complete rm 'n/*/f:^*.{c,h,cc,C,tex,1,man,l,y}/'
> ls
foo.c foo.o
> vi f[TAB]
> vi foo.c
> rm f[TAB]
> rm foo.o
```

Note that `fignore` is not used when a shell pattern is specified in a completion action.

Other commands don't expect files as arguments, so the completion builtin can be used to change the list of items to be completed:

```
> complete man     'n/*/c/' # complete commands
> complete which   'n/*/c/'
> complete dbx     'p/2/(core)/' 'n/*/c/'
> complete alias   'p/1/a/' # complete aliases
> complete set     'p/1/s/' # complete variables
> complete finger  'p/1/u/' # complete user-names
```

Note that in the `dbx` example, the positional completion is specified before the next word completion. Since completions are evaluated from left to right, if the next word completion was specified first it would always match and the positional completion would never be executed. This is a common mistake when defining a completion.

In some cases the list needs to be customized or dynamically generated.

```
> set hostnames=(prep.ai.mit.edu tesla.ee.cornell.edu)
> complete rlogin 'p/1/$hostnames/'
```

Consider a complex example using matching patterns to direct argument dependent completions:

```
> complete find 'n/-fstype/(nfs 4.2)/' 'n/-name/f/' \
  'n/-type/(c b d f p l s)/' 'n/-user/u/' 'n/-exec/c/' \
  'n/-ok/c/' 'n/-cpio/f/' 'n/-ncpio/f/' 'n/-newer/f/' \
  'c/-/(fstype name perm prune type user nouser \
    group nogroup size inum atime mtime ctime exec \
    ok print ls cpio ncpio newer xdev depth)/' \
  'n/*/d/'
```

The words following the `-fstype` and `-type` arguments is to be completed from the lists specified. The word following the `-name`, `-cpio`, `-ncpio`, and `-newer` arguments are to be completed using files or directory components. The word following the `-user` argument will be completed from the list of user names of the system, and the word following the `-exec` and *-ok* arguments will be completed from the list of commands. The fourth line specifies how the current word will be completed if it starts with a `-`. Note that the pattern of this completion argument refers to the current word and not to the previous word, like before.

Finally, the last line in the find completion matches all the remaining cases and requires a directory component.

```
> cd /usr
> set autolist
> find [TAB]
bin/ include/ lib/ local/ man/ src/
> find -[TAB]
atime   depth   group   mtime   newer    ok     prune   user
cpio    exec    inum    name    nogroup  perm   size    xdev
ctime   fstype  ls      ncpio   nouser   print  type
> find -u[TAB]
> find -user [TAB]
christos root
> find -user c[TAB]
> find -user christos
```

Another useful feature is to override the default suffix rules. For example:

```
> complete rcp 'c/*:/f/' 'p/1/$hosts/:'
> complete finger 'c/*@/$hosts/' 'p/1/u/@'
```

completes from $hosts appending a ':' and then begins expanding files, where finger expands user-names appending an '@' and then begins expanding hosts.

Another exotic example that makes use of dynamic command execution is:

```
> complete talk p/1/'`users | tr " " "\012" | uniq`'/ \
  n/\*/\`who\ \|\ grep\ \$:1\ \|\ awk\ \'\{\ print\ \$2\ \}\'\`/
```

Here we want to get a list of the active users, as an argument to talk, then we want to pick the user, and find his `tty`... Note that there is no quoting performed inside the completion patterns. If the character `/` is used in the matched pattern, then another punctuation character can be used as a separator. The only pattern recognized is `$:<number>` which is substituted by the n'th word of the current line.

Notes:

1. The completion command-patterns are overridden when the internal parsing mechanism determines that the completion is a login name, or a variable.
2. This is an experimental feature and the syntax/behavior may change in future versions.

FYI

This shell uses `CBREAK` mode but takes typed-ahead characters anyway. You can still use `stty` to set some of the modes of your terminal (but not bindings).

This shell will restore your `tty` to a sane mode if it appears to return from some command in raw, `CBREAK`, or `noecho` mode. This behavior can be changed using `setty`.

ENVIRONMENT

`HPATH`	Path to look for command documentation
`LANG`	Used to give preferred character environment (see NLS)
`LC_CTYPE`	Used to change only `ctype` character handling (see NLS)

NOREBIND	Inhibits rebinding of printable characters to self-insert-command
PATH	Path to look for command execution
SHLVL	Current shell level nesting
TERM	Used to tell how to handle the terminal
LINES	Number of lines in terminal (see WINDOW SIZE)
COLUMNS	Number of columns in terminal (see WINDOW SIZE)
TERMCAP	Terminal capability string (see WINDOW SIZE)
SYSTYPE	The current system type (Domain OS only)

NEW SHELL VARIABLES

addsuffix	add a / for directories, and a space for normal files when complete matches a name exactly. If unset don't add anything extra.
ampm	show all times in 12 hour, AM/PM format.
autocorrect	Correct mis-spelled path components automatically before attempting completion.
autoexpand	Invoke the expand-history function automatically on completion.
autolist	List possibilities on an ambiguous completion.
autologout	The first word indicates number of minutes of inactivity before automatic logout. The optional second word indicates the number of minutes of inactivity after which the user will be prompted for his login password to enter a command.
backslash_quote	Makes the backslash quote `\`, `'`, and `"`. This option changes the parsing mechanism for `tcsh`, and it can cause syntax errors in `csh` scripts.

`correct`	Automatically try to correct the spelling of commands. Must be set to either `correct=cmd`, only command name will be corrected, or `correct=all`, the whole line will be corrected.
`dextract`	Extract a directory on pushd rather than rotating.
`dunique`	Push only directories that are not already in the directory stack.
`echo_style`	Set the style of the builtin echo. Valid values are: `.RS "0.7in"`
`bsd :`	If the first argument is `-n`, don't echo a newline.
`sysv :`	Recognize \ escape sequences in echo strings.
`both :`	Recognize both `-n` and \ escape sequences.
`none :`	The purist's echo.
`edit`	Use the input editor, set by default.
`fignore`	List of file name suffixes (e.g. .o, ~) to ignore during complete.
`gid`	The current real group id.
`histlit`	If set, history lines in the editor will be shown with its literal value (that is the line as it was input) instead of the shells lexical version. The current history line can be toggled between literal and lexical with the toggle-literal-history function. History lines saved at shell exit are also saved as this variable indicates.
`histfile`	If set, it contains the full path-name where a history file is read/written. It defaults to `$home/.history`. This is useful when sharing the same home directory in different machines, or if one wants to save all the histories in the `tty` sessions. It is usually set in `.cshrc` for interactive shells, because history is sourced between `.cshrc` and `.login` so that it is available from `.login`.
`inputmode`	Can be set to either `insert` or `overwrite` to control the line editing behavior, as described in section 1.

`listjobs`	List all jobs when suspending. set `listjobs=long`, produces long format.
`listlinks`	Resolve symbolic links when listing files so that the correct filetype is shown.
`listmax`	Maximum number of items to list without asking first.
`matchbeep`	Control beeping on completion. With `matchbeep=nomatch`, completion only beeps when there is no match, with `matchbeep=ambiguous`, beeps also when there are multiple matches, with `matchbeep=notunique`, beeps when there is one exact and other longer matches, with `matchbeep=never`, it never beeps.
`nobeep`	Disables beeping completely.
`nokanji`	If `kanji` support is enable, setting this variable disables it, so that the meta key can be used.
`nostat`	An list of directories that should not be stat'ed during a completion operation. Some directories e.g. `/afs`, can take too much time to complete a `stat` operation, making them prime candidates for membership in this list.
`oid`	The organization id number (Domain OS only).
`printexitvalue`	If an interactive program exits non-zero, print the exit value.
`prompt`	The string to prompt with.
`prompt2`	The string to prompt for while and for loops with.
`prompt3`	The string to prompt with when automatic spelling correction has corrected a command line.
`pushdtohome`	Make `pushd` with no `args` do a `pushd ~` (like `cd` does).
`pushdsilent`	Do not print the `dir` stack on every `pushd` and `popd`.
`recexact`	Recognize exact matches even if they are ambiguous.

`recognize_only_executables`
: List choices of commands only displays files in the path that are executable (slow).

`rmstar`
: Prompt the user before execution of `rm*`.

`savedirs`
: Before exiting, create a file called `$HOME/.cshdirs` that contains the necessary commands to restore the current directory stack. `tsch` will read this file during startup and restore the directory stack.

`savehist`
: Number of history items to save between login sessions. If it is just set, the number of items is taken from `$history`.

`shlvl`
: Integer value indicating the number of nested shells.

`showdots`
: Show hidden files in list and complete operations. If it is set to `-A`, then `.` and `..` are not displayed, similarly to the BSD `ls -A`.

`symlinks`
: Can be set to: `chase` , in order to resolve the full path names, `ignore` , in order to use $cwd to resolve relative directory references for `builtins.` `expand` , in order to use `$cwd` and rewrite the arguments for commands.

`tcsh`
: Contains the current version of the shell as `R.VV.PP`. The `R` indicates the major release number, the `VV` the current version and the `PP` the patchlevel.

`term`
: The terminal type; see above.

`tperiod`
: Periodic command wait period (in minutes).

`tty`
: The name of the `tty`, or empty if not attached to one.

`uid`
: The current real user ID.

`version`
: The version ID stamp for this `tcsh`. It contains, the origin of this version of `tcsh`, the date this version was released and a string containing a comma separated list of the compile time options enabled:

  `8b 7b`
  : If `tcsh` was compiled to be eight bit clean or not. The default is `8b`.

`nls`	Set if `tcsh` uses the system's NLS, should be the default for systems that have NLS.
`lf`	Set if `tcsh` should execute `.login` before `.cshrc` on login shells. Default is not set.
`dl`	Set if `tcsh` should put `. last` on the path for security. Default is set.
`vi`	Set if `tcsh`'s default editor is vi. Default is unset (emacs)
`dtr`	Set if `tcsh` should drop `dtr` on login shells when exiting. Default is unset.
`bye`	Set if `tcsh` should accept `bye` in addition to logout, and rename log to watchlog. Default is unset.
`al`	Set if `tcsh` should determine if autologout should be enabled. The default is set.
`dir`	Set if `tcsh` should save and restore the directory stack.
`kan`	Set if `tcsh` is compiled for `Kanji`. (ignore the `iso` character set.) Default is unset.
`sm`	Set if `tcsh` was compiled to use the system's `malloc`.

In addition to the above strings, administrators can enter local strings to indicate differences in the local version.

`visiblebell` use the visible bell (screen flash) rather than audible bell.

`watch` List of events to watch.

`who` Format string for the printouts generated when `watch` is set.

`wordchars` List of non-alphanumeric characters considered part of a word for the purpose of the forward-word, backward-word etc. functions -- defaults to `*?_-.[]~=`.

NEW SPECIAL ALIASES

`cwdcmd`	The command is run after every change of working directory.
`periodic`	The command to be run every `tperiod` minutes.
`precmd`	The command to be run prior to printing each prompt.
`beepcmd`	The command to be run every time `tcsh` wants to echo the terminal bell.

SEE ALSO

Utility Program - `xterm, csh, chsh`

File Format - `termcap`

BUGS

The screen update for lines longer than the screen width is very poor if the terminal cannot move the cursor up (i.e. terminal type `dumb`).

I am certain that there are bugs. Bugs (preferably with fixes) should be sent to Christos Zoulas (`christos@ee.cornell.edu`).

VERSION

This man page documents `tcsh` 6.02.00 (Cornell) 05/15/92.

AUTHORS

Ken Greer, HP Labs, 1981 - Wrote the command completion.

Mike Ellis, Fairchild, 1983 - Added command name recognition/completion.

Paul Placeway, Ohio State CIS dept., 1983 - Added the command line editor.

Rayan Zachariassen, University of Toronto, 1984 - Added the builtin `which` feature to the editor, and the code for `ls-F` . Also numerous bug fixes, modifications, and performance enhancements.

Chris Kingsley, Caltech - Wrote the fast storage allocator routines (nmalloc.c).

Karl Kleinpaste, CCI 1983-4 - Added special aliases, directory stack extraction stuff, login/logout watch, and scheduled events. Also came up with the idea of the new prompt format.

Paul Placeway, Ohio State CIS dept., 1987 - Re-wrote the editor, cleaned up other code, and added the prompt routines, added to the syntax for file name expressions, and sped up the shell some.

Chris Grevstad, TRW, 1987 - Ported the 4.3 `csh` sources to `tcsh`.

Christos S. Zoulas, Cornell U. EE dept., 1987-91 - ported `tcsh` to HPUX, and System V rel. 2 and 3 and wrote a SysV version of `getwd.c`. Added SHORT_STRINGS support. New version of `sh.glob.c`.

James J Dempsey, BBN, 1988, and Paul Placeway, OSU, 1988 - Re-ported `tcsh` to A/UX.

BUGFIXES AND ENHANCEMENTS

Mika Liljeberg, liljeber@kruuna.Helsinki.FI - Linux port.

Scott Krotz, Motorola, 1991 - Minix port.

David Dawes, Sydney U. Australia, Physics dept., 1991 - SVR4 job control fixes. (reverse pipeline startup code, and signal fixes).

Jose Sousa, Interactive Systems Corp., 1991 extended vi fixes. Added delete command in `vi`.

Marc Horowitz, MIT, 1991 - Ansification fixes, new exec hashing code, imake fixes, where builtin.

Eric Schnoebelen, Convex, 1990 - Convex support, lots of `csh` bug fixes, save and restore of directory stack.

Jaap Vermeulen, Sequent, 1990-91 - Vi mode fixes, expand-line, window change fixes, ported to symmetry machines.

Ron Flax, Apple, 1990 - Ported again to A/UX 2.0

Dan Oscarsson, LTH Sweden, 1990 - NLS support and simulated for non NLS sites. Correction of file names also handles the case when the / is replaced by another character. The editor does not switch into cbreak mode unless needed. The shell will not use character attributes when output is not to a `tty`.

Johan Widen, SICS Sweden, 1990 - Shell level variable, mach support, correct-line, 8-bit printing.

Matt Day, Sanyo Icon, 1990 - Added POSIX termio support; Fixed limit stuff for SysV.

Hans J. Albertsson (Sun Sweden) - Added the ampm variable handling, DING!, and the `settc` and `telltc` builtins.

Michael Bloom Fixed some of the interrupt handling.

Michael Fine, Digital Equipment Corp added the extended key support.

Daniel Long, NNSC, 1988 added the wordchars variable.

George Hartzell, MCD Biology, University of Colorado-Boulder, 1988 fixed the always resetting to DEL bug.

Patrick Wolfe, Kuck and Associates, Inc., 1988 cleaned up VI mode and wrote the new editor description (in section 1).

Jak Kirman, 1988 fixed the SunOS 4 giant stack allocation bug.

Bruce Robertson, Tektronix, 1989 fixed setting erase and kill (again).

David C Lawrence, Rensselaer Polytechnic Institute, 1989 added `autolist` variable and code to list possibilities on ambiguous completion.

Martin Boyer, Institut de recherche d'Hydro-Quebec, 1991. Modified `autolist` variable and code to give options on beeping behavior. Modified the history search to search for the whole string from the beginning of the line to the cursor.

Alec Wolman, DEC, 1989 added code to allow newlines in the prompt.

Matt Landau, BBN, 1989 fixed YP bugs on Suns, added .tcshrc stuff.

Ray Moody, Purdue Physics, 1989 added the code to do magic spacebar history expansion.

Mordechai ????, Intel, 1989 re-arranged the `printprompt()` routine to use a switch(*cp) (rather than a bunch of ifs), and added a few things to it.

Josh Siegel, dspo.gov, 1989 fixed `fg-editor` and added the status line shell variables `sl` and `el`.

Karl Berry, UMB, 1989 fixed a bug involving environ (in sh.func.c) on NeXT computers.

Michael Greim, Universitaet des Saarlandes, Saarbruecken, W-Germany, fixed the nested backquote bug (yes, you can do it; think about aliases) in 4.2 `csh`. This fix was posted to comp.bugs.4bsd.

Kazuhiro Honda, Department of Computer Science, Keio University, 1989 added the code for automatic spelling correction, the prompt3 stuff (the autocorrect prompt), and `HOSTTYPE` symbols for the Sony NEWS.

Paul Placeway, BBN, 1990 fixed character redrawing code insert bugs, and made redrawing code handle multiple character movement, insert, and delete (if it exists). Fixed setting of SHELL.

Per Hedeland, Ellemtel, Sweden, 1990 - Various bugfixes and improvements, e.g. in history expansion, autolist, added autoexpand, updated the manual.

THANKS TO

A special thanks to: Bryan Dunlap, Clayton Elwell, Karl Kleinpaste, Bob Manson, Steve Romig, Diana Smetters, Bob Sutterfield, Mark Verber, Elizabeth Zwicky, and all the other people at Ohio State for suggestions and encouragement.

Also, thanks to all the people on the net for putting up with, reporting bugs in, and suggesting new additions to the old `tcsh` editor.

NAME

`tee` – distribute data to multiple files

SYNOPSIS

`tee [-ai]` *file ...*

DESCRIPTION

`tee` reads from the standard input and writes it to each file and to standard output. `tee` can create any number of output files at once. If more files than are available to one process are requested, `tee` forks a copy of itself to complete the list.

OPTIONS

`-a` Append output to the end of each file.

`-i` Ignore interrupts.

SEE ALSO

Utility Programs - `dlsh`

NAME

`telnet` – user interface to the TELNET protocol

SYNOPSIS

telnet [*host* **[** *port* **]]**

DESCRIPTION

`telnet` is used to communicate with another host using the TELNET protocol. If `telnet` is invoked without arguments, it enters command mode, indicated by its prompt (`telnet>`). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an `open` command (see below) with those arguments.

Once a connection has been opened, `telnet` enters an input mode. The input mode entered will be either character-at-a-time or line-by-line depending on what the remote system supports.

In character-at-a-time mode, most text typed is immediately sent to the remote host for processing.

In line-by-line mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The *local echo character* (initially `^E`) may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the `localchars` toggle is `TRUE` (the default in line mode; see below), the user's `quit`, `intr`, and `flush` characters are trapped locally, and sent as TELNET protocol sequences to the remote side. There are options (see `toggle autoflush` and `toggle autosynch` below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of `quit` and `intr`).

While connected to a remote host, `telnet` command mode may be entered by typing the `telnet escape character` (initially `^]`). When in command mode, the normal terminal editing conventions are available.

COMMANDS

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the `mode`, `set`, `toggle`, and `display` commands).

`open` *host* [*port*]
: Open a connection to the named host. If no port number is specified, `telnet` will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see `hosts(5)`) or an Internet address specified in the dot notation (see `inet(3N)`).

`close`
: Close a TELNET session and return to command mode.

`quit`
: Close any open TELNET session and exit `telnet`. An end of file (in command mode) will also close a session and exit.

`z`
: Suspend `telnet`. This command enters a sub-shell. The user may return to `telnet` by exiting the sub-shell.

`mode` *type*
: *Type* is either `line` (for line-by-line mode) or `character` (for character-at-a-time mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

`status`
: Show the current status of `telnet`. This includes the peer one is connected to, as well as the current mode.

`display` [a*rgument...*]
: Displays all, or some, of the `set` and `toggle` values (see below).

`?` [*command*]
: Get help. With no arguments, `telnet` prints a help summary. If a command is specified, `telnet` will print the help information for just that command.

`send` *arguments*
: Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

  `escape`
  : Sends the current `telnet` escape character (initially `^]`).

`synch`	Sends the `TELNET SYNCH` sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system—if it doesn't work, a lower case **`r`** may be echoed on the terminal).
`brk`	Sends the `TELNET BRK` (Break) sequence, which may have significance to the remote system.
`ip`	Sends the `TELNET IP` (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.
`ao`	Sends the `TELNET AO` (Abort Output) sequence, which should cause the remote system to flush all output from the remote system to the user's terminal.
`ayt`	Sends the `TELNET AYT` (Are You There) sequence, to which the remote system may or may not choose to respond.
`ec`	Sends the `TELNET EC` (Erase Character) sequence, which should cause the remote system to erase the last character entered.
`el`	Sends the `TELNET EL` (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.
`ga`	Sends the `TELNET GA` (Go Ahead) sequence, which likely has no significance to the remote system.
`nop`	Sends the `TELNET NOP` (No Operation) sequence.
?	Prints out help information for the send command.

`set` *argument value*

Set any one of a number of `telnet` variables to a specific value. The special value "off" turns off the function associated with the variable. The values of variables may be interrogated with the display command. The variables which may be specified are:

`echo` This is the value (initially `^E`) which, when in line-by-line mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

`escape` This is the `telnet` escape character (initially `^[`) which causes entry into `telnet` command mode (when connected to a remote system).

`interrupt`
If `telnet` is in `localchars` mode (see `toggle localchars` below) and the `interrupt` character is typed, a `TELNET IP` sequence (see `send ip` above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's `intr` character.

`quit` If `telnet` is in `localchars` mode (see `toggle localchars` below) and the `quit` character is typed, a `TELNET BRK` sequence (see `send brk` above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's `quit` character.

`flushoutput`
If `telnet` is in `localchars` mode (see `toggle localchars` below) and the `flushoutput` character is typed, a `TELNET AO` sequence (see `send ao` above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's `flush` character.

`erase` If `telnet` is in `localchars` mode (see `toggle localchars` below), and if `telnet` is operating in character-at-a-time mode, then when this character is typed, a `TELNET EC` sequence (see `send ec` above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's `erase` character.

`kill` If `telnet` is in `localchars` mode (see `toggle localchars` below), and if `telnet` is operating in character-at-a-time mode, then when this character is typed, a `TELNET EL` sequence (see `send el` above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's `kill` character.

`eof` If `telnet` is operating in line-by-line mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the `eof` character is taken to be the terminal's `eof` character.

`toggle` *arguments...*
Toggle (between `TRUE` and `FALSE`) various flags that control how `telnet` responds to events. More than one argument may be specified. The state of these flags may be interrogated with the `display` command. Valid arguments are:

`localchars`
If this is `TRUE`, then the `flush`, `interrupt`, `quit`, `erase`, and `kill` characters (see `set` above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively `ao`, `ip`, `brk`, `ec`, and `el`; see `send` above). The initial value for this toggle is `TRUE` in line-by-line mode, and `FALSE` in character-at-a-time mode.

`autoflush`
: If `autoflush` and `localchars` are both `TRUE`, then when the `ao`, `intr`, or `quit` characters are recognized (and transformed into TELNET sequences; see set above for details), `telnet` refuses to display any data on the user's terminal until the remote system acknowledges (via a `TELNET Timing Mark` option) that it has processed those TELNET sequences. The initial value for this toggle is `TRUE` if the terminal user had not done an `stty noflsh`, otherwise `FALSE` (see `stty(1)`).

`autosynch`
: If `autosynch` and `localchars` are both `TRUE`, then when either the `intr` or `quit` characters is typed (see `set`, above, for descriptions of the `intr` and `quit` characters), the resulting TELNET sequence sent is followed by the `TELNET SYNCH` sequence. This procedure *should* cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is `FALSE`.

`crmod`
: Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is `FALSE`.

`debug`
: Toggles socket level debugging (useful only to the super user). The initial value for this toggle is `FALSE`.

`options`	Toggles the display of some internal `telnet` protocol processing (having to do with TELNET options). The initial value for this toggle is `FALSE`.
`netdata`	Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is `FALSE`.
`?`	Displays the legal `toggle` commands.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

FILES

`/etc/hosts`	
`/etc/hosts.equiv`	(on remote host)
`/etc/inetd.conf`	
`$HOME/.rhosts`	(on remote host).

SEE ALSO

Utility Programs - `inetd, telnetd`

File Formats - `hosts, inetd.conf`

NAME

`telnetd` – DARPA TELNET protocol server

SYNOPSIS

`telnetd`

DESCRIPTION

`telnetd` is a server which supports the DARPA standard TELNET virtual terminal protocol. `telnetd` is invoked by the internet server (see `inetd`), normally for requests to connect to the TELNET port as indicated by the `/etc/services` file (see `services`).

`telnetd` operates by allocating a pseudo-terminal device (see `pty`) for a client, then creating a login process which has the slave side of the pseudo-terminal as `stdin`, `stdout`, and `stderr`. `telnetd` manipulates the master side of the pseudo-terminal, implementing the TELNET protocol and passing characters between the remote client and the login process.

When a TELNET session is started up, `telnetd` sends TELNET options to the client side indicating a willingness to do `remote echo` of characters, to `suppress go ahead`, and to receive `terminal type information` from the remote client. If the remote client is willing, the remote terminal type is propagated in the environment of the created login process. The pseudo-terminal allocated to the client is configured to operate in cooked mode, and with `XTABS` and `CRMOD` enabled (see `tty`).

`telnetd` is willing to do: `echo`, `binary`, `suppress go ahead`, and `timing mark`. `telnetd` is willing to have the remote client do: `binary`, `terminal type`, and `suppress go ahead`.

NOTE

`telnetd` never sends TELNET go-ahead commands.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `telnet, inetd`

File Formats - `inetd.conf`

NAME

`termcap` – retrieve `termcap` data

SYNOPSIS

`termcap` *type entry col row*

DESCRIPTION

`termcap` looks up requested *entry* in the terminal information database `/etc/termcap` and prints the values on the standard output. The type of terminal is determined by examining the `TERM` environment variable.

OPTIONS

The *type* of the *entry* is indicated by one of the following characters.

f	Flag entry, such as `am` (automatic margin). `termcap` prints `T` if the flag is set; `F` if not.
n	Numeric entry, such as `li` (lines on the screen).
s	String entry, such as `ce` (clear to end of line).
c	Cursor positioning entry, such as `cm`. For this type, `termcap` creates the output string using the value of the field combined with the *row* and *col* information given.

If `termcap` is invoked as `clear` (via a hard link), it behaves as if it had been invoked as `termcap s cl`. The display of the terminal is cleared.

EXAMPLES

To clear the screen, the command

```
termcap s cl
```

will suffice (although `clear` is simpler), while

```
$#set(lines, "termcap n li")
```

sets a shell variable to the number of lines on the screen.

SEE ALSO

Utility Programs - `clear, tset`

Library Functions - `termcap()`

File Formats - `termcap`

NAME

`test` – simple relational expression evaluator

SYNOPSIS

`test` *expression*

DESCRIPTION

`test` evaluates a relational *expression* according to the syntax described below and then exits with a return code to indicate truth (0) or falsehood (1).

EXPRESSION SYNTAX

Each separate argument to `test` is taken to be an independent component of the expression. Therefore, all operators must be separated from operands by one or more spaces when `test` is invoked from a shell.

Operators are listed in order of decreasing precedence.

`– pred` *string* — Examines *string* in one of several ways and either succeeds or fails. Success results in the numeric result 1; failure is 0.

- **`-d`** — Succeeds if *string* is the name of a directory.
- **`-l`** — Succeeds if *string* is the name of a symbolic link.
- **`-f`** — Succeeds if *string* is the name of a regular file (i.e., not a directory or special file or symbolic link).
- **`-w`** — Succeeds if *string* is the name of an existing file which is writable.
- **`-r`** — Succeeds if *string* is the name of an existing file which is readable.
- **`-s`** — Succeeds if *string* is the name of a non-empty existing file.

	-t	Succeeds if *string*, interpreted as a decimal number, represents a file descriptor which is open and connected to a terminal. For this predicate, *string* can be omitted; if it is, file descriptor one is assumed.
	-z	Succeeds if *string* is the null *string*.
	-n	Succeeds if *string* is not the null *string*.

x **-eq** *y*	Succeeds if integer *x* is equal to integer *y*. A leading minus sign on an integer is allowable to indicate a negative value.
x **-ne** *y*	Succeeds if integer *x* is not equal to integer *y*.
x **-gt** *y*	Succeeds if integer *x* is greater than integer *y*.
x **-ge** *y*	Succeeds if integer *x* is greater than or equal to integer *y*.
x **-lt** *y*	Succeeds if integer *x* is less than integer *y*.
x **-le** *y*	Succeeds if integer *x* is less than or equal to integer *y*.
string	Succeeds if *string* is not the null *string*; identical to **-n** *string*.
str1 = *str2*	Succeeds if string *str1* is identical to string *str2*.
str1 **!=** *str2*	Succeeds if string *str1* is not equal to string *str2*.
! *expr*	Succeeds if *expr* fails.
e1 **-a** *e2*	Succeeds if expression *e1* and expression *e2* both succeed.
e1 **-o** *e2*	Succeeds if either of the expressions *e1* or *e2* succeed.

Parentheses may be used for grouping. Like the other operators, they must be given to `test` as distinct command line arguments.

SEE ALSO

Utility Programs - `expr`

NAME

`tftp` – trivial file transfer program

SYNOPSIS

`tftp [` *host* **`]`**

DESCRIPTION

`tftp` is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case `tftp` uses *host* as the default host for future transfers (see the `connect` command below).

COMMANDS

Once `tftp` is running, it issues the prompt `tftp>` and recognizes the following commands:

`connect` *host_name* [*port*]
: Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers. Thus, the `connect` command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the `connect` command; the remote host can be specified as part of the `get` or `put` commands.

`mode` *transfer_mode*
: Set the mode for transfers; *transfer_mode* may be one of ASCII or `binary`. The default is ASCII.

`put` *file*
`put` *localfile remotefile*
`put` *file1 file2 ... fileN remote_directory*
: Put a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a file name on the remote host, if the host has already been specified, or a string of the form `host:filename` to

specify both a host and file name at the same time. If the latter form is used, the hostname specified becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a UNIX machine.

`get` *filename*
`get` *remotename localname*
`get` *file1 file2 ... fileN*
Get a file or set of files from the specified sources. `Source` can be in one of two forms: a file name on the remote host, if the host has already been specified, or a string of the form `host:filename` to specify both a host and file name at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.

`quit` Exit `tftp`. An end of file also exits.

`verbose` Toggle verbose mode.

`trace` Toggle packet tracing.

`status` Show current status.

`rexmt` *retransmission_timeout*
Set the per-packet retransmission timeout, in seconds.

`timeout` *total_transmission_timeout*
Set the total transmission timeout, in seconds.

`ascii` Shorthand for "mode ascii"

`binary` Shorthand for "mode binary"

`?` [*command-name ...*]
Print help information.

NOTE

Because there is no user-login or validation within the TFTP protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site and therefore difficult to document here. Specifically, the file should exist and should have write permission for `others` for the `put` command to work.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `inetd, tftpd`

File Formats - `inetd.conf`

NAME

`tftpd` – DARPA Trivial File Transfer Protocol server

SYNOPSIS

`tftpd`

DESCRIPTION

`tftpd` is a server which supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the `tftp` service description; see `services`. The server is normally started by `inetd`.

The use of `tftp` does not require an account or password on the remote system. Due to the lack of authentication information, `tftpd` will allow only publicly readable files to be accessed. Files may be written only if they already exist and are publicly writable. Note that this extends the concept of “public” to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling `tftp` service. The server should have the user ID with the lowest possible privilege.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `tftp, inetd`

File Formats - `inetd.conf`

NAME

`tic` – terminfo compiler

SYNOPSIS

tic [-?vct] [-o *pathname***] [-n** *ttyname***] [** *sourcefile***]**

DESCRIPTION

`tic` compiles a terminfo or termcap source file into compiled `terminfo` format files. The generated files are placed in the directory `/usr/lib/terminfo`.

OPTIONS

-? Displays help for all options.

-v Verbose. `tic` prints the information as the source file is compiled.

-c Check only *sourcefile* for errors, and do not create the compiled `terminfo` files. Errors in `use=`*ttyname* (`tc=`*ttyname*) references are not detected.

-t `tic` generally expects the input source file to be in `terminfo` format. The **-t** option tells `tic` to expect the input source file in the older `termcap` format. The `termcap` format differs from the `terminfo` source format in the keywords for each capability and in the delimiters used between capabilities.

-o *pathname* *pathname* is used instead of `/usr/lib/terminfo` as the directory for the compiled files and for `use=`*ttyname* (`tc=`*ttyname*) processing. The **-o** option takes precedence over the effects of the environment variable `TERMINFO` (see below).

-n *ttyname* Compile only the files associated with `ttyname` and all its aliases in the input source file.

sourcefile	If specified, *sourcefile* will be taken as the source file. Otherwise `terminfo.src` is the source file for terminfo format source, and `termcap.src` is the source file for `termcap` format source (**-t** option).

NOTES

When `tic` encounters the reference `use=`*ttyname* (`tc=`*ttyname*), it reads in the compiled file for terminal *ttyname* from the directory `/usr/lib/terminfo` (or *pathname* if the **-o** option is given; or `$TERMINFO` if set and the **-o** option is not given) to complete the current terminal entry. `tic` copies from the *ttyname* file only those capabilities which are not already defined in the current terminal entry.

ENVIRONMENT VARIABLES

`TERMINFO`	If set, its value is taken as the directory to use instead of `/usr/lib/terminfo` the compiled files and for `use=`*ttyname* (`tc=`*ttyname*) processing. The **-o** option takes precedence over the `TERMINFO` environment variable.

SEE ALSO

Utility Programs -	`termcap`
Library Functions -	`termcap()`
File Formats -	`termcap`

NAME

`time` – show a command's elapsed time

SYNOPSIS

`time [-v]` *program arg ...*

DESCRIPTION

`time` executes *program* and records the amount of system resources used. The environment variable `PATH` is consulted to locate the program if the pathname is not absolute. Each argument is passed to the program as if the program were being executed by the shell.

`time` produces the following statistics:

u	User time, as minutes:seconds.
s	System time.
e	Elapsed (real) time.
%	Percent of total time.

OPTIONS

-v	Verbose mode; long descriptions of each field are printed.

NOTE

The real time elapsed is accurate to within one second, and the CPU times (elapsed user and system times) are accurate to within 1/100 of a second.

SEE ALSO

Utility Programs - `dlsh, prof`

System Calls - `getrusage(), gettimeofday()`

NAME

`torom` – convert executable to put in ROM

DESCRIPTION

`torom` converts any `a.out` format executable into multiple hexadecimal format files required for burning into ROM.

Please refer to the platform-specific manual page for information on the `torom` command. The `torom` manual page for 68k systems is `torom_68k`; the `torom` manual page for x86 systems is `torom_x86`. For example:

```
man torom_x86
```

This will display the manual page for the x86-specific `torom` command.

NAME

`torom` – convert executable to put in ROM (68k only)

SYNOPSIS

torom *image_file eprom_size***[k] [0|8|16] [main|debug|opt] [flat] [raw|bin|intel] [swap]**

DESCRIPTION

`torom` converts any `a.out` format executable into multiple hexadecimal format files required for burning into ROM.

OPTIONS

image_file — The promable image file. This typically is either the ROMable output file of `mkimage` utility or the concatenated file of `bootcode` and `tftpcode` to facilitate network booting.

eprom_size — The size of EPROM in bytes. Suffix **k** can be used to interpret the size in kilobytes. `torom` will use this size information and accordingly create output files bounded by this size.

0 | 8 | 16 — Data split size. This split size determines where to split data for high & low EPROMs.

0	No data splitting; just a continuous byte stream.
8	8 bit byte split.
16	16 bit word split (the default).

main|debug|opt — This option indicates whether the PROMs will be put in the main slot with a 512KB space allocated for a debugger, or whether it will be put in the optional slot. Usually for MVME systems, the bug monitor resides in main slot and optional slot is available for LynxOS. The default is the optional slot.

`flat`	The **`flat`** option prevents `torom` from adding boot header and checksum. This is sometimes useful during testing.
`raw|bin|intel`	This option specifies the output format.

- `raw:` Output file in raw binary format (a byte stream).
- bin: Output file in formatted binary format; includes a header and data.
- intel: Output file in Intel MCS-86 format (the default).

`swap` If this option is specified, bytes will be swapped in the output file. Use this option only with 16 bit data split.

EXAMPLE

Examples of typical usage:

on MVME167 `torom image_file 512k 16`

on MVME162 `torom image_file 512k 0`

SEE ALSO

Utility programs - `mkimage, mkramdisk`

File Formats - `mkimage.spec`

NAME

`torom` – convert executable to put in ROM (x86 only)

SYNOPSIS

`torom` *image_file eprom_size***`[k] [main/opt]`**

DESCRIPTION

`torom` converts any `a.out` format executable into multiple hexadecimal format files required for burning into ROM.

The *eprom_size* value can be suffixed with **`k`** to indicate size in number of kilobytes.

OPTIONS

`-?`	Displays help message of all available options.
image_file	The promable image file. This typically is either the ROMable output file of `mkimage` utility or the concatenated file of `bootcode` and `tftpcode` to facilitate network booting. It may also be the small preboot code that transfers control to the ROMable kernel image on power-up or system reset.
eprom_size	The size in bytes of EPROMs to be used in the target system. Accordingly, `torom` creates multiple output files with names, `xx.low` and `xx.high` appended to the image file. Here `xx` indicates the sequence numbers of the PROM sets.

SEE ALSO

Utility Programs - `mkimage`

NAME

`touch` – change the modify date of files

SYNOPSIS

`touch [-cf]` *file ...*

DESCRIPTION

`touch` sets the modification date of a file to the current date. If the file exists, the date is changed by reading a character from it and then writing it back. If the file does not exist, it is created.

OPTIONS

`-c` If the file does not exist, do not create it.

`-f` Perform a `touch` even if read and write permissions are denied on the file.

SEE ALSO

Utility Programs - `ls, mkdir`

System Calls - `stat()`

NAME

`tr` – translate or delete characters

SYNOPSIS

`tr [-cst] [--complement] [--squeeze-repeats]`
`[--truncate-set1]` *string1 string2*

`tr {-s,--squeeze-repeats} [-c] [--complement]` *string1*

`tr {-d,--delete} [-c]` *string1*

`tr {-d,--delete} {-s,--squeeze-repeats} [-c] [--complement]`
string1 string2

`tr --help`

`tr --version`

OPTIONS

`-c` Forms a virtual *string1* argument as the ASCII collating sequence minus the characters given in *string1*.

`-d,--delete` Deletes any occurrences of *string1* characters in the input stream.

`-s,--squeeze-repeats`
Removes contiguous occurrences of *string1* characters from the input stream, leaving just one occurrence in their place.

`-t` Truncates the effective *string1* argument to the length of *string2*.

`--help` Print a usage message and exit with a status code indicating success.

`--version` Print version information on standard output then exit.

DESCRIPTION

This manual page documents the GNU version of `tr`. `tr` copies the standard input to the standard output, performing one of the following operations:

- translate, and optionally squeeze repeated characters in the result
- squeeze repeated characters
- delete characters
- delete characters, then squeeze repeated characters from the result.

The *string1* and (if given) *string2* arguments define ordered sets of characters, referred to below as set1 and set2. These sets are the characters of the input that `tr` operates on. The **`--complement`** (**`-c`**) option replaces set1 with its complement (all of the characters that are not in set1).

Specifying Sets of Characters

The format of the *string1* and *string2* arguments resembles the format of regular expressions; however, they are not regular expressions, only lists of characters. Most characters simply represent themselves in these strings, but the strings can contain the shorthands listed below, for convenience. Some of them can be used only in *string1* or *string2*, as noted below.

Backslash escapes. A backslash followed by a character not listed below causes an error message.

`\a`	Control-G.
`\b`	Control-H.
`\f`	Control-L.
`\n`	Control-J.
`\r`	Control-M.
`\t`	Control-I.

`\v`	Control-K.
`\ooo`	The character with the value given by *ooo*, which is 1 to 3 octal digits.
`\\`	A backslash.

Ranges

The notation '*m*–*n*' expands to all of the characters from *m* through *n*, in ascending order. *m* should collate before *n*; if it doesn't, an error results. As an example, '0–9' is the same as '0123456789'. Although GNU `tr` does not support the System V syntax that uses square brackets to enclose ranges, translations specified in that format will still work as long as the brackets in string1 correspond to identical brackets in string2.

Repeated characters

The notation '`[`*c*`*`*n*`]`' in *string2* expands to *n* copies of character *c*. Thus, '[y*6]' is the same as 'yyyyyy'. The notation '`[`*c*`*]`' in *string2* expands to as many copies of *c* as are needed to make set2 as long as set1. If *n* begins with a 0, it is interpreted in octal, otherwise in decimal.

Character classes

The notation '`[:`*class-name*`:]`' expands to all of the characters in the (predefined) class named *class-name*. The characters expand in no particular order, except for the '`upper`' and '`lower`' classes, which expand in ascending order. When the **`--delete`** (**`–d`**) and **`--squeeze-repeats`** (**`–s`**) options are both given, any character class can be used in *string2*. Otherwise, only the character classes '`lower`' and '`upper`' are accepted in *string2*, and then only if the corresponding character class ('upper' and 'lower', respectively) is specified in the same relative position in *string1*. Doing this specifies case conversion. The class names are given below; an error results when an invalid class name is given.

`alnum`	Letters and digits.
`alpha`	Letters.
`blank`	Horizontal whitespace.

`cntrl`	Control characters.
`digit`	Digits.
`graph`	Printable characters, not including space.
`lower`	Lowercase letters.
`print`	Printable characters, including space.
`punct`	Punctuation characters.
`space`	Horizontal or vertical whitespace.
`upper`	Uppercase letters.
`xdigit`	Hexadecimal digits.

Equivalence classes

The syntax '`[=`*c*`=]`' expands to all of the characters that are equivalent to *c*, in no particular order. Equivalence classes are a recent invention intended to support non-English alphabets. But there seems to be no standard way to define them or determine their contents. Therefore, they are not fully implemented in Lynx GNU `tr`; each character's equivalence class consists only of that character, which makes this a useless construction currently.

Translating

`tr` performs translation when *string1* and *string2* are both given and the **`--delete`** (**–d**) option is not given. `tr` translates each character of its input that is in set1 to the corresponding character in set2. Characters not in set1 are passed through unchanged. When a character appears more than once in set1 and the corresponding characters in set2 are not all the same, only the final one is used. For example, these two commands are equivalent:

```
tr aaa xyz
tr a z
```

A common use of `tr` is to convert lowercase characters to uppercase. This can be done in many ways. Here are three of them:

```
tr abcdefghijklmnopqrstuvwxyz \
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
tr a-z A-Z
```

```
tr '[:lower:]' '[:upper:]'
```

When `tr` is performing translation, set1 and set2 should normally have the same length. If set1 is shorter than set2, the extra characters at the end of set2 are ignored.

On the other hand, making set1 longer than set2 is not portable; POSIX.2 says that the result is undefined. In this situation, the BSD `tr` pads set2 to the length of set1 by repeating the last character of set2 as many times as necessary. The System V `tr` truncates set1 to the length of set2.

By default, Lynx GNU `tr` handles this case like the BSD `tr` does. When the **--truncate-set1** (**-t**) option is given, GNU `tr` handles this case like the System V `tr` instead. This option is ignored for operations other than translation.

Acting like the System V `tr` in this case breaks the relatively common BSD idiom:

```
tr -cs A-Za-z0-9 '\012'
```

because it converts only zero bytes (the first element in the complement of set1), rather than all non-alphanumerics, to newlines.

Squeezing Repeats and Deleting

When given just the **--delete** (**-d**) option, `tr` removes any input characters that are in set1.

When given just the **--squeeze-repeats** (**-s**) option, `tr` replaces each input sequence of a repeated character that is in set1 with a single occurrence of that character.

When given both the **--delete** and the **--squeeze-repeats** options, `tr` first performs any deletions using set1, then squeezes repeats from any remaining characters using set2.

The **--squeeze-repeats** option may also be used when translating, in which case `tr` first performs translation, then squeezes repeats from any remaining characters using set2.

Here are some examples to illustrate various combinations of options:

Remove all zero bytes:

```
tr -d '\000'
```

Put all words on lines by themselves. This converts all non-alphanumeric characters to newlines, then squeezes each string of repeated newlines into a single newline:

```
tr -cs '[a-zA-Z0-9]' '[\n*]'
```

Convert each sequence of repeated newlines to a single newline:

```
tr -s '\n'
```

Warning Messages

Setting the environment variable `POSIXLY_CORRECT` turns off several warning and error messages, for strict compliance with POSIX.2. The messages normally occur in the following circumstances:

- When the **--delete** option is given but **--squeeze-repeats** is not, and *string2* is given, Lynx GNU `tr` by default prints a usage message and exits, because *string2* would not be used. The POSIX specification says that *string2* must be ignored in this case. Silently ignoring arguments is a bad idea.
- When an ambiguous octal escape is given. For example, `\400` is actually `\40` followed by the digit 0, because the value `400` octal does not fit into a single byte.

OTHER COMPATIBILITY ISSUES

Note that GNU `tr` does not provide complete BSD or System V compatibility. For example, there is no option to disable interpretation of the POSIX constructs `[:alpha:]`, `[=c=]`, and `[c*10]`. Also, Lynx GNU `tr` does not delete zero bytes automatically, unlike traditional UNIX versions, which provide no way to preserve zero bytes.

NAME

`trace_net` – Enables or disables logging mode for `routed` daemon

SYNOPSIS

`trace_net on` *filepath machine...*

`trace_net off` *machine...*

DESCRIPTION

`trace_net` sends messages to request that the logging of route messages either be enabled or disabled. If enabling them, specify the log file as *filepath*. To capture route messages in the `/usr/tmp/route.log` file on a system named `lynxmachine`, enter:

```
trace_net on /usr/tmp/route.log lynxmachine
```

To disable any further logging of route messages, enter:

```
trace_net off lynxmachine
```

SEE ALSO

Utility Programs - `routed`

NAME

`true, false` – provide truth values

SYNOPSIS

`true`

`false`

DESCRIPTION

`true` and `false` do nothing, and return 0 and 1, respectively. They are useful in writing Bourne shell scripts.

EXAMPLES

```
while true
do
  command list
done
```

SEE ALSO

Utility Programs - `sh`

NAME

`tset` – set up terminal line

SYNOPSIS

`tset [-dv] [-t` *type*] **`[-c`** *config*]

`tset -S`

`tset -T`

DESCRIPTION

`tset` quickly initializes the line parameters and software features for a terminal. In the first form, `tset` consults the `/etc/termcap` file to find initialization strings and soft tab setup strings, and the `/etc/tconfig` file for line initialization data. The terminal type and configuration type are normally taken from the environment. If desired, the actual `termcap` entry for the terminal type is printed to standard output.

In the second and third forms, `tset` simply prints the current line speed or the current terminal type.

OPTIONS

`-d` Print the `termcap` entry for the current terminal type. This information is normally stored into the environment variable `TERMCAP`.

`-v` Print error messages to the standard error channel if any initializations go awry. Normally, errors are silently ignored.

`-t` *type* Use the given *type* as the terminal type instead of the type found in the environment variable `TERM`. The terminal type is used to locate initialization information in the `/etc/termcap`

`-c` *config* Use the given *config* type instead of the configuration type found in the environment variable CONFIG. The configuration type is used to locate line setup information in the file `/etc/tconfig`.

-S The current line speed (in baud) is printed. No other action is taken.

-T The current terminal type is printed. No other action is taken. Useful if the `TERM` environment variable is incorrect.

SEE ALSO

Utility Programs - `stty`

System Calls - `ioctl()`

Library Functions - `ttyconfig()`

NAME

`tty` – print name of current `tty` device

SYNOPSIS

`tty`

DESCRIPTION

`tty` prints the pathname of the `tty` device associated with the file descriptor open on channel 0. If the device is not a `tty`, the string `not a tty` is printed.

NOTE

The directory `/dev` is the only one examined for pathnames.

SEE ALSO

Utility Programs - `tset`

Library Functions - `ttyname()`

NAME

umount – unmount a file system

SYNOPSIS

umount *device_name*

umount *dir*

umount [–vq] –a

umount [–vq] –m

umount –h *host*

umount –t *type*

DESCRIPTION

umount makes the system remove a mounted file system from the specified directory *dir*. umount can also be invoked to unmount all mounted file systems from a specific host or to unmount all mounted filesystems of NFS or BLOCK type. In either case, the umount refers to the file /etc/fstab to get the mounted file system information.

OPTIONS

special	Removes the file system previously mounted on *device_name* and updates /etc/mtab.
dir	Unmounts the file system or the remote NFS directory previously mounted under the directory *dir* and updates /etc/mtab.
–a	Unmount all file systems listed in /etc/fstab.
–m	Unmount all file systems listed in /etc/mtab.
–v	Print the directory name as the corresponding file system is unmounted.
–q	Unmount file system but do not update /etc/mtab.

-t *type*	Unmount entries of matching type from `/etc/fstab`. Supported *type*(s) are NFS and block.
-h *host*	Unmount all NFS mounted directories from the server *host*.

EXAMPLES

The following examples show the functions of `umount` with different options when called from the command line shell prompt.

Unmount all file system entries mentioned in `/etc/fstab`.

umount -a

Unmount the mounted file system from /usr/dl directory

umount /usr/dl

Unmount all NFS mounts from the host "topcat" that are in `/etc/fstab`

umount -h topcat

Umount all block device mounts mentioned in `/etc/fstab`

umount -t block

Umount all NFS mount entries mentioned in `/etc/fstab`

umount -t NFS

Unmount all entries based on `/etc/mtab`

umount -m

FILES

`/etc/mtab`	Mount table.
`/etc/fstab`	Filesystem table.

SEE ALSO

Utility Programs - `mount`

System Calls - `mount()`, `umount()`

NAME

`uname` – return system name.

SYNOPSIS

`uname [-amnrsv]`

DESCRIPTION

The `uname` utility writes the current system name to standard output. The format and contents of the system names are those supported by the `uname()` function.

OPTIONS

The following options are available:

-a	The `uname` utility behaves as though all the options (**-mnrsv**) were specified.
-m	Writes the hardware machine name to standard output.
-n	Writes the system node name to standard output. This is an alias that the system uses for intersystem communication addressing.
-r	Write the operating system release level to standard output.
-s	Write the operating system name to standard output.
-v	Write the operating system version to standard output.

If no options are selected, `uname` prints the system name, as if **-s** option had been specified.

EXIT STATUS

The `uname` utility exits with the following status:

0	Upon successful completion.
>0	For syntax errors in specifying options.

NAME

`unfsio` – start network input/output daemon (NFS Client)

SYNOPSIS

unfsio [*timeout***] [***count***]**

DESCRIPTION

`unfsio` runs in the background and communicates with kernel tasks using a special character device `/dev/unfs` to handle requests for NFS Client support. It should be started after networking is initialized and before trying to mount remote NFS directories.

Without arguments, `unfsio` uses a default timeout value of 1 second and default retry count of 86,400 for `RPC` requests to remote systems. The *timeout* parameter specifies the RPC timeout value in milliseconds and *count* specifies the number of times the RPC request is retried before returning with error status to the kernel. In such cases, the **`errno`** variable in the user is set to `EIO` error value.

The `unfsio` command may be given multiple number of times to run multiple `unfsio` tasks simultaneously. This will allow for multiple NFS requests being serviced simultaneously to improve NFS throughput. The limit on the number of simultaneous `unfsio` tasks can be selected by changing the `unfs_info` structure in the `/sys/devices/nfsinfo.c` file.

NOTE

If the NFS server is down and `unfsio` is invoked using the timeout and retry value defaults (to simulate hardmount), `unfsio` will continue retrying the client user process request for up to 24 hours; during which time the waiting user process cannot be interrupted. The only way to interrupt the user process is to kill the `unfsio` process and restart it with a smaller number of retries (using command line arguments) to implement a softmount.

FILES

`/dev/unfs`	Character device to process NFS requests.
`/sys/devices/nfsinfo.c`	NFS Client configuration parameters.

NAME

`uniq` – remove adjacent duplicate lines

SYNOPSIS

uniq [-cdu [+*number***|-***number***]] [***infile* **[***outfile***]]**

DESCRIPTION

`uniq` is used to get non-duplicated lines out of a file that is in sorted order and which is suspected of having duplicate, adjacent lines in it. `uniq` only works on a sorted file; you can use `sort` to achieve this.

If no arguments are given, `uniq` compares the entire lines, and outputs a copy of the input without duplicated adjacent lines.

OPTIONS

-c Prepend a count of how many times each line occurred to the lines printed by `uniq`.

-d Print out a single copy of just the lines which had duplicates.

-u Print out only those lines which did not have duplicates.

(Note `uniq`'s default behavior is the combination of the **-d** and **-u** options).

-*number* Skip the first *number* fields in each line before beginning the comparison. A field is a string of non-white-space characters, separated from other fields by white-space characters (space and tab are the white-space characters)

+*number* Skip the first *number* characters in each line before beginning the comparison. You can combine **-***number* and **+***number*; `uniq` skips fields, then characters.

SEE ALSO

Utility Programs - `sort`

NAME

uuencode, uudecode – encode/decode a binary file

SYNOPSIS

uuencode [*file* **]** *name*

uudecode [*file* **]** *...*

DESCRIPTION

uuencode and uudecode are used to transmit binary files over transmission mediums that do not support other than simple ASCII data. uuencode reads file (or by default the standard input) and writes an encoded version to the standard output. The encoding uses only printing ASCII characters and includes the mode of the file and the operand name for use by uudecode.

uudecode transforms uuencoded files (or by default, the standard input) into the original form. The resulting file is named name and will have the mode of the original file except that setuid and execute bits are not retained. uudecode ignores any leading and trailing lines.

EXAMPLES

The following example packages up a source tree, compresses it, uuencodes it and mails it to a user on another system. When uudecode is run on the target system, the file src_tree.tar.Z will be created which may then be uncompressed and extracted into the original tree.

```
tar cf - src_tree &| compress &|
uuencode src_tree.tar.Z &| mail sys1!sys2!user
```

SEE ALSO

Utility Programs - compress, mail, uucp

File Formats - uuencode

BUGS

The encoded form of the file is expanded by 35% (3 bytes become 4 plus control information).

HISTORY

The `uuencode` command appeared in BSD 4.0.

COPYRIGHTS

NAME

`vi, ex, view` – text editors

SYNOPSIS

ex [-eFRrsv] [-c *cmd*] **[-t** *tag*] **[-w** *size*] [*file* ...]
vi [-eFlRrv] [-c *cmd*] **[-t** *tag*] **[-w** *size*] [*file* ...]
view [-eFRrv] [-c *cmd*] **[-t** *tag*] **[-w** *size*] [*file* ...]

DESCRIPTION

`vi` is a screen oriented text editor. `ex` is a line-oriented text editor. `ex` and `vi` are different interfaces to the same program, and it is possible to switch back and forth during an edit session. `view` is the equivalent of using the **-R** (read-only) option of `vi`.

This manual page is the one provided with the `nex/nvi` versions of the `ex/vi` text editors. `nex/nvi` are intended as bug-for-bug compatible replacements for the original Fourth Berkeley Software Distribution (4BSD) `ex` and `vi` programs. For the rest of this manual page, `nex/nvi` is used only when it's necessary to distinguish it from the historic implementations of `ex/vi`.

This manual page is intended for users already familiar with `ex/vi`. Anyone else should almost certainly read a good tutorial on the editor before this manual page. If you're in an unfamiliar environment, and you absolutely have to get work done immediately, read the section after the options description, entitled "Fast Startup". It's probably enough to get you going.

The following options are available:

-c *cmd*
: Execute *cmd* immediately after starting the edit session. Particularly useful for initial positioning in the file, however *cmd* is not limited to positioning commands. This is the POSIX 1003.2 interface for the historic "+*cmd*" syntax. `nex/nvi` supports both the old and new syntax.

-e
: Start editing in ex mode, as if the command name were `ex`.

-F	Don't copy the entire file when first starting to edit. (The default is to make a copy in case someone else modifies the file during your edit session.)
-l	Start editing with the lisp and showmatch options set.
-R	Start editing in read-only mode, as if the command name was `view`, or the `readonly` option was set.
-r	Recover the specified files, or, if no files are specified, list the files that could be recovered. If no recoverable files by the specified name exist, the file is edited as if the **-r** option had not been specified.
-s	Enter batch mode; applicable only to `ex` edit sessions. Batch mode is useful when running `ex` scripts. Prompts, informative messages and other user oriented message are turned off, and no startup files or environmental variables are read. This is the POSIX 1003.2 interface for the historic "**-**" argument. `nex/nvi` supports both the old and new syntax.
-t *tag*	Start editing at the specified *tag*. (See `ctags`(1)).
-w *windowlines*	Set the initial window size to the specified number of lines.
-v	Start editing in vi mode, as if the command name was `vi` or `view`.

Command input for `ex/vi` is read from the standard input. In the `vi` interface, it is an error if standard input is not a terminal. In the `ex` interface, if standard input is not a terminal, `ex` will read commands from it regardless, however, the session will be a batch mode session, exactly as if the **-s** option had been specified.

`ex/vi` exits 0 on success, and greater than 0 if an error occurs.

FAST STARTUP

This section will tell you the minimum amount that you need to do simple editing tasks using `vi`. If you've never used any screen editor before, you're likely to have problems even with this simple introduction. In that case you should find someone that already knows `vi` and have them walk you through this section.

`vi` is a screen editor. This means that it takes up almost the entire screen, displaying part of the file on each screen line, except for the last line of the screen. The last line of the screen is used for you to give commands to `vi`, and for `vi` to give information to you.

The other fact that you need to understand is that `vi` is a modeful editor, i.e. you are either entering text or you are executing commands, and you have to be in the right mode to do one or the other. You will be in command mode when you first start editing a file. There are commands that switch you into input mode. There is only one key that takes you out of input mode, and that is the <escape> key. (Key names are written using less-than and greater-than signs, e.g. <escape> means the "escape" key, usually labeled "esc" on your terminal's keyboard.) If you're ever confused as to which mode you're in, keep entering the <escape> key until `vi` beeps at you. (Generally, `vi` will beep at you if you try and do something that's not allowed. It will also display error messages.)

To start editing a file, enter the command "vi file_name<carriage-return>". The command you should enter as soon as you start editing is ":set verbose showmode<carriage-return>". This will make the editor give you verbose error messages and display the current mode at the bottom of the screen.

The commands to move around the file are:

`h` Move the cursor left one character.

`j` Move the cursor down one line.

`k` Move the cursor up one line.

`l` Move the cursor right one character.

`<cursor-arrows>`
The cursor arrow keys should work, too.

/*text*<carriage-return>
: Search for the string *text* in the file, and move the cursor to its first character.

The commands to enter new text are:

a	Append new text, *after* the cursor.
i	Insert new text, *before* the cursor.
o	Open a new line below the line the cursor is on, and start entering text.
O	Open a new line above the line the cursor is on, and start entering text.
<escape>	Once you've entered input mode using the one of the a, i, O or o commands, use <escape> to quit entering text and return to command mode.

The commands to copy text are:

yy	Copy the line the cursor is on.
p	Append the copied line after the line the cursor is on.

The commands to delete text are:

dd	Delete the line the cursor is on.
x	Delete the character the cursor is on.

The commands to write the file are:

:w<carriage-return>
: Write the file back to the file with the name that you originally used as an argument on the vi command line.

:w *file_name*<carriage-return>
: Write the file back to the file with the name "*file_name*".

The commands to quit editing and exit the editor are:

`:q<carriage-return>`
: Quit editing and leave vi (if you've modified the file, but not saved your changes, `vi` will refuse to quit).

`:q!<carriage-return>`
: Quit, discarding any modifications that you may have made.

One final caution. Unusual characters can take up more than one column on the screen, and long lines can take up more than a single screen line. The above commands work on "physical" characters and lines, i.e. they affect the entire line no matter how many screen lines it takes up and the entire character no matter how many screen columns it takes up.

VI COMMANDS

The following section describes the commands available in the command mode of the `vi` editor. In each entry below, the tag line is a usage synopsis for the command character.

[*count*] `<control-A>`
: Search forward *count* times for the current word.

[*count*] `<control-B>`
: Page backwards *count* screens.

[*count*] `<control-D>`
: Scroll forward *count* lines.

[*count*] `<control-E>`
: Scroll forward *count* lines, leaving the current line and column as is, if possible.

[*count*] `<control-F>`
: Page forward *count* screens.

`<control-G>` Display the file information.

`<control-H>`
[*count*] `h` Move the cursor back *count* characters in the current line.

[*count*] `<control-J>`
[*count*] `<control-N>`
[*count*] `j` Move the cursor down *count* lines without changing the current column.

`<control-L>`
`<control-R>` Repaint the screen.

[*count*] `<control-M>`
[*count*] `+` Move the cursor down *count* lines to the first nonblank character of that line.

[*count*] `<control-P>`
[*count*] `k` Move the cursor up *count* lines, without changing the current column.

`<control-T>` Return to the most recent tag context.

`<control-U>` Scroll backwards *count* lines.

`<control-W>` Switch to the next lower screen in the window, or, to the first screen if there are no lower screens in the window.

`<control-Y>` Scroll backwards *count* lines, leaving the current line and column as is, if possible.

`<control-Z>` Suspend the current editor session.

`<escape>` Execute `ex` commands or cancel partial commands.

`<control-]>` Push a tag reference onto the tag stack.

`<control-^>` Switch to the most recently edited file.

[*count*] `<space>`
[*count*] `l` Move the cursor forward *count* characters without changing the current line.

[*count*] `!` *motion* `shell-argument(s)`
Replace text with results from a shell command.

[*count*]`# #|+|-` Increment or decrement the cursor number.

[*count*] `$` Move the cursor to the end of a line.

`%` Move to the matching character.

`&`	Repeat the previous substitution command on the current line.
`'`*<character>* `` ` ``*<character>*	Return to a context marked by the character *<character>*.
`[`*count*`] (`	Back up *count* sentences.
`[`*count*`] )`	Move forward *count* sentences.
`[`*count*`] ,`	Reverse find character *count* times.
`[`*count*`] -`	Move to first nonblank of the previous line, *count* times.
`[`*count*`] .`	Repeat the last `vi` command that modified text.

`/RE<carriage-return>`
`/RE/ [offset]<carriage-return>`
`?RE<carriage-return>`
`?RE? [`*offset*`]<carriage-return>`

Search forward(`/`*RE*`/`) or backward(`?`*RE*`?`) for regular expression *RE*, positioning cursor *offset* lines beyond a matched line. The RE is saved for use with `n` commands, which are interpreted as "find next occurrence" operations.

`N`	Display the next file in the sequence of edit *file*s.
`n`	Search forward or backward for a regular expression.
`0`	Move to the first character in the current line.
`:`	Execute an ex command.
`[`*count*`] ;`	Repeat the last character find *count* times.
`[`*count*`]<` *motion* `[`*count*`]>` *motion*	Shift lines left or right.
`@` *buffer*	Execute *buffer*.
`[`*count*`] A`	Enter input mode, appending the text after the end of the line.
`[`*count*`] B`	Move backwards *count* bigwords.

[*buffer*] [*count*] `C`	Change text from the current position to the end-of-line.
[*buffer*] `D`	Delete text from the current position to the end-of-line.
[*count*] `E`	Move forward *count* end-of-bigwords.
[*count*] `F` *<character>*	Search *count* times backward through the current line for *<character>*.
[*count*] `G`	Move to line *count*, or the last line of the file if *count* not specified.
[*count*] `H`	Move to the screen line *count* – 1 lines below the top of the screen.
[*count*] `I`	Enter input mode, inserting the text at the beginning of the line.
[*count*] `J`	Join lines.
[*count*] `L`	Move to the screen line *count* – 1 lines above the bottom of the screen.
`M`	Move to the screen line in the middle of the screen.
[*count*] `O`	Enter input mode, appending text in a new line above the current line.
[*buffer*] `P`	Insert text from a *buffer*.
`Q`	Exit `vi` (or visual) mode and switch to `ex` mode.
[*count*] `R`	Enter input mode, replacing the characters in the current line.
[*buffer*] [*count*] `S`	Substitute *count* lines.
[*count*] `T` *<character>*	Search backwards, *count* times, through the current line for the character *after* the specified *<character>*.
`U`	Restore the current line to its state before the cursor last moved to it.
[*count*] `W`	Move forward *count* bigwords.

`[`*buffer*`] [`*count*`] X`
: Delete *count* characters before the cursor.

`[`*buffer*`] [`*count*`] Y`
: Copy (or "yank") *count* lines into the specified *buffer*.

`ZZ` Write the file and exit `vi`.

`[`*count*`] [[` Back up *count* section boundaries.

`[`*count*`] ]]` Move forward *count* section boundaries.

`^` Move to first nonblank character on the current line.

`[`*count*`] _` Move down *count* – 1 lines, to the first nonblank character.

`[`*count*`] a` Enter input mode, appending the text after the cursor.

`[`*count*`] b` Move backwards *count* words.

`[`*buffer*`] [`*count*`] c` *motion*
: Change a region of text.

`[`*buffer*`] [`*count*`] d` *motion*
: Delete a region of text.

`[`*count*`] e` Move forward *count* end-of-words.

`[`*count*`] f`*<character>*
: Search forward, *count* times, through the rest of the current line for *<character>*.

`[`*count*`] i` Enter input mode, inserting the text before the cursor.

`m` *<character>* Save the current context (line and column) as *<character>*.

`[`*count*`] o` Enter input mode, appending text in a new line under the current line.

`[`*buffer*`] p` Append text from a *buffer*.

`[`*count*`] r` *<character>*
: Replace *count* characters.

[*buffer*] [*count*] s
: Substitute *count* characters in the current line starting with the current character.

[*count*] t <*character*>
: Search forward, *count* times, through the current line for the character immediately *before* <*character*>.

u
: Undo the last change made to the file.

[*count*] w
: Move forward *count* words.

[*buffer*] [*count*] x
: Delete *count* characters.

[*buffer*] [*count*] y *motion*
: Copy (or “yank”) a text region specified by the *count* and *motion* into a *buffer*.

[*count1*] z [*count2*] -|.|+|^|<carriage-return>
: Redraw, optionally repositioning and resizing the screen.

[*count*] {
: Move backward *count* paragraphs.

[*count*] |
: Move to a specific *column* position on the current line.

[*count*] }
: Move forward *count* paragraphs.

[*count*]~
: Reverse the case of the next *count* character(s).

[*count*]~ *motion*
: Reverse the case of the characters in a text region specified by the *count* and *motion*.

<interrupt>
: Interrupt the current operation.

VI TEXT INPUT COMMANDS

The following section describes the commands available in the text input mode of the vi editor.

<nul>
: Replay the previous input.

<control-D>
: Erase to the previous shiftwidth column boundary.

`^<control-D>`	Erase all of the autoindent characters, and reset the autoindent level.
`0<control-D>`	Erase all of the autoindent characters.
`<control-T>`	Insert sufficient *<tab>* and *<space>* characters to move forward to the next `shiftwidth` column boundary.
`<erase>` `<control-H>`	Erase the last character.
`<literal next>`	Quote the next character.
`<escape>`	Resolve all text input into the file, and return to command mode.
`<line erase>`	Erase the current line.
`<control-W>` `<word erase>`	Erase the last word. The definition of word is dependent on the `altwerase` and `ttywerase` options.
`<control-X>[0-9A-Fa-f]+`	Insert a character with the specified hexadecimal value into the text.
`<interrupt>`	Interrupt text input mode, returning to command mode.

EX COMMANDS

The following section describes the commands available in the `ex` editor. In each entry below, the tag line is a usage synopsis for the command.

`<end-of-file>`

Scroll the screen.

`! argument(s)`
`[`*range*`]! argument(s)`

Execute a shell command, or filter lines through a shell command.

`"` A comment.

[*range*] nu[mber] [*count*] [*flags*]
[*range*] # [*count*] [*flags*]
: Display the selected lines, each preceded with its line number.

@ *buffer*
* *buffer* : Execute a *buffer*.

[*line*] a[ppend][!]
: The input text is appended after the specified *line*.

[*range*] c[hange][!] [*count*]
: The input text replaces the specified *range*.

cs[cope] add | find | help | kill | reset
: Execute a Cscope command.

[*range*] d[elete] [*buffer*] [*count*] [*flags*]
: Delete the lines from the file.

di[splay] b[uffers] | c[onnections] | s[creens] | t[ags]
: Display *buffer*s, Cscope connections, screens or tags.

[Ee][dit][!] [+*cmd*] [*file*]
[Ee]x[!] [+*cmd*] [*file*]
: Edit a different file.

exu[sage] [command]
: Display usage for an ex command.

f[ile] [file]
: Display and optionally change the file name.

[Ff]g [name] : vi mode only. Foreground the specified screen.

[*range*] g[lobal] /*pattern*/ [*commands*]
[*range*] v /*pattern*/ [*commands*]
: Apply *commands* to lines matching (or not matching) a *pattern*.

he[lp] : Display a help message.

[*line*] i[nsert][!]
: The input text is inserted before the specified *line*.

[*range*] `j[oin][!]` [*count*] [*flags*]
: Join lines of text together.

[*range*] `l[ist]` [*count*] [*flags*]
: Display the lines unambiguously.

`map[!] [lhs rhs]`
: Define or display maps (for `vi` only).

[*line*] `ma[rk]` *<character>*
[*line*] `k` *<character>*
: Mark the *line* with the mark *<character>*.

[*range*] `m[ove]` *line*
: Move the specified lines after the target *line*.

`mk[exrc][!]` *file*
: Write the abbreviations, editor options and maps to the specified *file*.

`[Nn][ext][!]` [*file* ...]
: Edit the next *file* from the argument list.

[*line*] `o[pen]` /*pattern*/ [*flags*]
: Enter open mode.

`pre[serve]`
: Save the file in a form that can later be recovered using the `ex` **-r** option.

`[Pp]rev[ious][!]`
: Edit the previous file from the argument list.

[*range*] `p[rint]` [*count*] [*flags*]
: Display the specified lines.

[*line*] `pu[t]` [*buffer*]
: Append *buffer* contents to the current line.

`q[uit][!]`
: End the editing session.

[*line*] `r[ead][!]` [*file*]
: Read a *file*.

`rec[over]` *file*
: Recover *file* if it was previously saved.

`res[ize] [+|-]`*size*
: `vi` mode only. Grow or shrink the current screen.

`rew[ind][!]` Rewind the argument list.

`se[t] [option[=[value]] ...]`
`set [nooption ...]`
`set [option? ...]`
`set [all]` Display or set editor options.

`sh[ell]` Run a shell program.

`so[urce]` *file* Read and execute `ex` commands from a *file*.

`[`*range*`] s[ubstitute] [/`*pattern*`/`*replace*`/][`*options*`][`*count*`][`*flags*`]`
`[`*range*`] &[`*options*`][`*count*`][`*flags*`]`
`[`*range*`] ~ [`*options*`] [`*count*`] [`*flags*`]`
Make substitutions.

`su[spend][!]`
`st[op][!]`
`<suspend>` Suspend the edit session.

`[Tt]a[g][!]` *tagstring*
Edit the file containing the specified tag.

`tagn[ext][!]` Edit the file containing the next context for the current tag.

`tagp[op][!] [`*file* | *number*`]`
Pop to the specified tag in the tags stack.

`tagp[rev][!]` Edit the file containing the previous context for the current tag.

`unm[ap][!] lhs`
Unmap a mapped string.

`ve[rsion]` Display the version of the `ex/vi` editor.

`[`*line*`] vi[sual] [`*type*`] [`*count*`] [`*flags*`]`
`ex` mode only. Enter `vi`.

`[Vi]i[sual][!] [+`*cmd*`] [`*file*`]`
`vi` mode only. Edit a new *file*.

`viu[sage] [command]`
Display usage for a `vi` command.

[*range*] `w[rite][!] [>>]` [*file*]
[*range*] `w[rite] [!]` [*file*]
[*range*] `wn[!] [>>]` [*file*]
[*range*] `wq[!] [>>]` [*file*]
Write the *file*.

[*range*] `x[it][!] [file]`
Write the file if it has been modified.

[*range*] `ya[nk]` [*buffer*] [*count*]
Copy the specified lines to a *buffer*.

[*line*] `z` [*type*] [*count*] [*flags*]
Adjust the window.

SET OPTIONS

There are a large number of options that may be set (or unset) to change the editor's behavior. This section describes the options, their abbreviations and their default values.

In each entry below, the first part of the tag line is the full name of the option, followed by any equivalent abbreviations. The part in square brackets is the default value of the option. Most of the options are boolean, i.e. they are either on or off, and do not have an associated value.

Options apply to both `ex` and `vi` modes, unless otherwise specified.

`altwerase [off]`
`vi` only. Select an alternate word erase algorithm.

`autoindent, ai [off]`
Automatically indent new lines.

`autoprint, ap [off]`
`Ex` only. Display the current line automatically.

`autowrite, aw [off]`
Write modified files automatically when changing files.

`backup` Backup files before they are overwritten.

`beautify, bf [off]`
Discard control characters.

`cdpath [environment variable CDPATH, or current directory]`
The directory paths used as path prefixes for the `cd` command.

`cedit [no default]`
Set the character to edit the colon command-line history.

`columns, co [80]`
Set the number of columns in the screen.

`comment [off]` `vi` only. Skip leading comments in shell and C-language files.

`directory, dir [environment variable TMPDIR, or /tmp]`
The directory where temporary files are created.

`edcompatible, ed [off]`
Remember the values of the “c” and “g” suffices to the `substitute` commands, instead of initializing them as unset for each new command.

`errorbells, eb [off]`
`ex` only. Announce error messages with a bell.

`exrc, ex [off]` Read the startup files in the local directory.

`extended [off]` Regular expressions are extended (i.e. `egrep`-style) expressions.

`filec [no default]`
Set the character to perform file path completion on the colon command line.

`flash [on]` Flash the screen instead of beeping the keyboard on error.

`hardtabs, ht [8]`
Set the spacing between hardware tab settings.

`iclower [off]` Makes all Regular Expressions case-insensitive, as long as an upper-case letter does not appear in the search string.

`ignorecase, ic [off]`
Ignore case differences in regular expressions.

`keytime [6]` The 10th's of a second `ex/vi` waits for a subsequent key to complete a key mapping.

`leftright [off]`
`vi` only. Do left-right scrolling.

`lines, li [24]` `vi` only. Set the number of lines in the screen.

`lisp [off]` `vi` only. Modify various search commands and options to work with Lisp. *This option is not yet implemented.*

`list [off]` Display lines in an unambiguous fashion.

`lock [on]` Attempt to get an exclusive lock on any file being edited, read or written.

`magic [on]` Treat certain characters specially in regular expressions.

`matchtime [7]` `vi` only. The 10th's of a second `ex/vi` pauses on the matching character when the `showmatch` option is set.

`mesg [on]` Permit messages from other users.

`modelines, modeline [off]`
Read the first and last few lines of each file for `ex` commands. *This option will never be implemented.*

`noprint` Characters that are never handled as printable characters.

`number, nu [off]`
Precede each line displayed with its current line number.

`octal [off]` Display unknown characters as octal numbers, instead of the default hexadecimal.

`open [on]` `ex` only. If this option is not set, the `open` and `visual` commands are disallowed.

`optimize, opt [on]`
`vi` only. Optimize text throughput to dumb terminals. *This option is not yet implemented.*

`paragraphs, para [IPLPPPQPP LIpplpipbp]`
: `vi` only. Define additional paragraph boundaries for the `{` and `}` commands.

`printp`
: Characters that are always handled as printable characters.

`prompt [on]`
: `Ex` only. Display a command prompt.

`readonly, ro [off]`
: Mark the file and session as read-only.

`recdir [/var/tmp/vi.recover]`
: The directory where recovery files are stored.

`redraw, re [off]`
: `vi` only. Simulate an intelligent terminal on a dumb one. *This option is not yet implemented.*

`remap [on]`
: Remap keys until resolved.

`report [5]`
: Set the number of lines about which the editor reports changes or yanks.

`ruler [off]`
: `vi` only. Display a row/column ruler on the colon command line.

`scroll, scr [window / 2]`
: Set the number of lines scrolled.

`searchincr [off]`
: Makes the `/` and `?` commands incremental.

`sections, sect [NHSHH HUnhsh]`
: `vi` only. Define additional section boundaries for the `[[` and `]]` commands.

`secure [off]`
: Turns off all access to external programs.

`shell, sh [environment variable SHELL, or /bin/sh]`
: Select the shell used by the editor.

`shellmetaQ`
: Set the meta characters checked to determine if file name expansion is necessary.

`shiftwidth, sw [8]`
Set the autoindent and shift command indentation width.

`showmatch, sm [off]`
`vi` only. Note matching "{" and "(" for "}" and ")" characters.

`showmode, smd [off]`
`vi` only. Display the current editor mode and a "modified" flag.

`sidescroll [16]`
`vi` only. Set the amount a left-right scroll will shift.

`slowopen, slow [off]`
Delay display updating during text input. *This option is not yet implemented.*

`sourceany [off]`
Read startup files not owned by the current user. *This option will never be implemented.*

`tabstop, ts [8]` This option sets tab widths for the editor display.

`taglength, tl [0]`
Set the number of significant characters in tag names.

`tags, tag [tags /var/db/libc.tags /sys/kern/tags]`
Set the list of tags files.

`term, ttytype, tty [environment variable TERM]`
Set the terminal type.

`terse [off]` This option has historically made editor messages less verbose. It has no effect in this implementation.

`tildeop [off]` Modify the ~ command to take an associated motion.

`timeout, to [on]`
Time out on keys which may be mapped.

`ttywerase [off]`
`vi` only. Select an alternate erase algorithm.

`verbose [off]` `vi` only. Display an error message for every error.

`w300 [no default]`
: `vi` only. Set the window size if the baud rate is less than 1200 baud.

`w1200 [no default]`
: `vi` only. Set the window size if the baud rate is equal to 1200 baud.

`w9600 [no default]`
: `vi` only. Set the window size if the baud rate is greater than 1200 baud.

`warn [on]`
: `ex` only. This option causes a warning message to the terminal if the file has been modified, since it was last written, before a `!` command.

`window, w, wi [environment variable LINES]`
: Set the window size for the screen.

`windowname [off]`
: Change the icon/window name to the current file name even if it can't be restored on editor exit.

`wraplen, wl [0]`
: `vi` only. Break lines automatically, the specified number of columns from the left-hand margin. If both the `wraplen` and `wrapmargin` edit options are set, the `wrapmargin` value is used.

`wrapmargin, wm [0]`
: `vi` only. Break lines automatically, the specified number of columns from the right-hand margin. If both the `wraplen` and `wrapmargin` edit options are set, the `wrapmargin` value is used.

`wrapscan, ws [on]`
: Set searches to wrap around the end or beginning of the file.

`writeany, wa [off]`
: Turn off file-overwriting checks.

ENVIRONMENTAL VARIABLES

`COLUMNS`	The number of columns on the screen. This value overrides any system or terminal specific values. If the `COLUMNS` environmental variable is not set when `ex/vi` runs, or the `columns` option is explicitly reset by the user, `ex/vi` enters the value into the environment.
`EXINIT`	A list of `ex` startup commands, read if the variable `NEXINIT` is not set.
`HOME`	The user's home directory, used as the initial directory path for the startup "$`HOME/.nexrc`" and "$`HOME/.exrc`" files. This value is also used as the default directory for the `vi cd` command.
`LINES`	The number of rows on the screen. This value overrides any system or terminal specific values. If the `LINES` environmental variable is not set when `ex/vi` runs, or the `lines` option is explicitly reset by the user, `ex/vi` enters the value into the environment.
`NEXINIT`	A list of `ex` startup commands.
`SHELL`	The user's shell of choice (see also the `shell` option).
`TERM`	The user's terminal type. The default is the type "unknown". If the `TERM` environmental variable is not set when `ex/vi` runs, or the `term` option is explicitly reset by the user, `ex/vi` enters the value into the environment.
`TMPDIR`	The location used to stored temporary files (see also the `directory` edit option).

ASYNCHRONOUS EVENTS

`SIGALRM`	`vi/ex` uses this signal for periodic backups of file modifications and to display "busy" messages when operations are likely to take a long time.
`SIGHUP`	

`SIGTERM`	If the current buffer has changed since it was last written in its entirety, the editor attempts to save the modified file so it can be later recovered. See the `vi/ex` Reference manual section entitled “Recovery” for more information.
`SIGINT`	When an interrupt occurs, the current operation is halted, and the editor returns to the command level. If interrupted during text input, the text already input is resolved into the file as if the text input had been normally terminated.
`SIGWINCH`	The screen is resized. See the `vi/ex` Reference manual section entitled “Sizing the Screen” for more information.
`SIGCONT`	
`SIGQUIT`	
`SIGTSTP`	`vi/ex` ignores these signals.

FILES

`/bin/sh`	The default user shell.
`/etc/vi.exrc`	System-wide vi startup file.
`/tmp`	Temporary file directory.
`/var/tmp/vi.recover`	The default recovery file directory.
`$HOME/.nexrc`	1st choice for user’s home directory startup file.
`$HOME/.exrc`	2nd choice for user’s home directory startup file.
`.nexrc`	1st choice for local directory startup file.

`.exrc`	2nd choice for local directory startup file.

SEE ALSO

Utility Programs - `ctags`

Library Functions - `more`, `curses`, `dbopen`

Articles - "An Introduction to Display Editing with Vi", found in the *UNIX User's Manual Supplementary Documents* section of both the 4.3BSD and 4.4BSD manual sets. This document is the closest thing available to an introduction to the `vi` screen editor.

"Ex Reference Manual (Version 3.7)", found in the *UNIX User's Manual Supplementary Documents* section of both the 4.3BSD and 4.4BSD manual sets. This document is the final reference for the `ex` editor, as distributed in most historic 4BSD and System V systems.

"Edit: A tutorial", found in the *UNIX User's Manual Supplementary Documents* section of the 4.3BSD manual set. This document is an introduction to a simple version of the `ex` screen editor.

"Ex/Vi Reference Manual", found in the *UNIX User's Manual Supplementary Documents* section of the 4.4BSD manual set. This document is the final reference for the `nex/nvi` text editors, as distributed in 4.4BSD and 4.4BSD-Lite.

HISTORY

The `nex/nvi` replacements for the `ex/vi` editor first appeared in 4.4BSD.

STANDARDS

`nex/nvi` is close to IEEE Std1003.2 ("POSIX"). That document differs from historical `ex/vi` practice in several places; there are changes to be made on both sides.

NAME

`vmstart` – start the virtual memory management

SYNOPSIS

`vmstart` *swap_name* **[***size***]**

`vmstart` **–** *size*

DESCRIPTION

`vmstart` uses the `vmstart` system call to start the virtual memory management and demand paging using the file *swap_name* with the given *size* or by paging from application text when **–** is specified instead of *swap_name*. If *size* is not mentioned, then *size* of the file *swap_name* is determined and used. Note that when you wish to use paging from application text, you must specify the *size* of virtual memory.

The *size* value can be suffixed with **`m/M`**, in which case it is interpreted as a number of megabytes. Similarly, the suffix **`k/K`** signifies kilobytes, and **`b/B`** signifies the number of 512-byte blocks. If none of the above suffixes is mentioned, the default is *size* bytes.

`vmstart` will fail if your swap space is smaller then physical memory; i.e., you have 8MB of main memory and run `vmstart` with a swap space of 4MB.

The size of virtual memory is not physical memory size + swap space size. Once you engage virtual memory, the new memory size becomes the swap space size. So, if you have 8MB of physical memory and a swap space of 12MB, you get 12MB of total memory not 20.

Once `vmstart` is started programs may or may not be swapped depending on the priority they are started with. The decision to swap or not is made at load time by the kernel. All programs run before `vmstart` is run will not swap. A program started at a priority less then or equal to the start priority of `vmstart` will always be able to swap. A program started at a priority higher then the start priority of `vmstart` will never swap. This means if a program tries to run at a higher priority then `vmstart` was started with may not have enough real memory to run.

NOTE

The start priority of `vmstart` is located in `/bin/rc`.

DIAGNOSTICS

`vmstart` fails under the following conditions:

If the *swap-name* specified cannot be used as a swap space.

If the size of the swap space is less then the size of real memory.

If the user issuing the command is not a superuser.

If the virtual memory has earlier been started by this command.

SEE ALSO

System Calls - `vmstart(), fstat()`

NAME

wc – count words, lines, characters in a file

SYNOPSIS

wc [-clw] [*file ...* **]**

DESCRIPTION

wc counts the number of words, characters, and lines in each *file* and output the results according to the options selected. All options are assumed if none are given. If no files are named, then the standard input is assumed to be the source of input. A word is a string delimited by spaces, tabs, or newlines. A line is a sequence of characters delimited by a newline character.

If only the character count is requested, wc obtains this information without scanning all the input (if possible), and is thus in this case very fast.

OPTIONS

-c	Output character count.
-l	Output line count.
-w	Output word count.

SEE ALSO

Utility Programs - du, ls

System Calls - stat()

NAME

`whatis` – search the `whatis` database for complete words

SYNOPSIS

`whatis` *keyword* ...

DESCRIPTION

`whatis` searches a set of database files containing short descriptions of system commands for keywords and displays the result on the standard output. Only complete word matches are displayed.

The `whatis` database is created using the command `/usr/lib/makewhatis`.

SEE ALSO

Utility Programs - `apropos`, `man`

NAME

`which` – locate a command

SYNOPSIS

which [**-i**] [**-a**] [**--**] [*command*]

DESCRIPTION

`which` provides the user with full expansion of the *command* argument, be it either an alias, a shell function, or an executable file (default). To enable search for aliases and shell functions the user should supply the **-i** (= interactive) flag. In that case `which` expects as standard input the expansion of the alias or shell function. If the standard input is empty, or the **-i** flag has not been given, `which` will try to locate *command* in the user's `PATH`. The interactive mode is easily used by setting an alias like the following in `csh`:

```
alias which alias !\$ \| /bin/which -i !\*
```

or

```
alias which eval alias '\"\$$#\" |' \
/bin/which -i '${1+"$@"}'
```

in shells which are supersets of `sh` and which know aliases. If your shell has shell functions, you can use the following function:

```
which()
{
  eval last=\"\$$#\"
  set | sed -n "/^$last(){$/,/^}$/p" |
  /bin/which -i ${1+"$@"}
}
```

If the **-a** (= all) flag is given, `which` will not stop after the first "match", but will search for all occurrences of *command* in the user's `PATH`**.** The **--** (double hyphen) flag can be used to end the list of options. The next argument (if present) will be taken as *command*, even if it starts with a **-** (single hyphen). `which` [**-i**] [**-a**] [**--**] without further arguments prints the user's `PATH`, broken up into its components, one per line.

This new version of the `which` command is not a `csh` script. Being an executable it is much faster; not sourcing `.cshrc`, it gives a true picture of one's aliases. Furthermore, it will give the correct answers even if:

– the "effective" `uid` (`gid`) differs from the "real" `uid` (`gid`)

– the effective `uid` is `0` (`root`).

EXAMPLE

```
% alias
which alias !$ | /usr/local/bin/which -i !*

% which which
which alias !$ | /usr/local/bin/which -i !*

% which -a which
which alias !$ | /usr/local/bin/which -i !*
/usr/local/bin/which
/usr/ucb/which
%
```

AUTHOR

Maarten Litmaath @ VU Informatika Amsterdam

DIAGNOSTICS

A diagnostic is given if *command* is not found.

SEE ALSO

Utility Program - `csh`

NAME

`who` – show current users

SYNOPSIS

`who [`*utmpfile***`]`**

`who am i`

`whoami`

DESCRIPTION

`who` reads a record of logins and reports the user names, times, and terminal device names associated with each. By default, `who` reads the file `/etc/utmp` which contains entries for each user current using the system Another file, most notably `/usr/adm/wtmp`, can be used by specifying utmpfile. The file `/usr/adm/wtmp` contains an ever-growing record of logins, logouts, and system crashes. Lines representing logouts have no user names. Lines containing tilde (~) characters contain times at which the system was brought down or restarted.

When invoked with two arguments, or as `whoami`, `who` prints the name of the current user.

SEE ALSO

Utility Programs - `ps`

File Formats - `utmp`

NAME

`wish` – Simple windowing shell

SYNOPSIS

wish [*options***[***arg***]]***...*

OPTIONS

-display *display*
: Display (and screen) on which to display window.

-file *fileName*
: Read commands from *fileName* rather than standard input. The last element in *fileName* will be used as the title for the application and name of its interpreter for `send` commands (unless overridden by the **-name** option).

-geometry *geometry*
: Initial geometry to use for window.

-name *name*
: Use *name* as the title to be displayed in the window, and as the name of the interpreter for `send` commands.

-sync
: Execute all X server commands synchronously, so that errors are reported immediately. This will result in much slower execution, but it is useful for debugging.

-help
: Print a summary of the command-line options and exit.

DESCRIPTION

`wish` is a simple program consisting of the Tcl command language, the Tk toolkit, and a main program that reads commands from standard input or from a file. It creates a main window and then processes Tcl commands. When `wish` is invoked without the **-f** option, it reads Tcl commands interactively from standard input. It will continue processing commands until all windows have been deleted or until end-of-file is reached on standard input.

If there exists a file `.wishrc` in the home directory of the user, `wish` evaluates the file as a Tcl script just before reading the first command from standard input.

If the **`-file`** option is provided to Tk, then `wish` reads Tcl commands from the file named in the **`-file`** option. These commands will normally create an interactive interface consisting of one or more widgets. When the end of the command file is reached, `wish` will continue to respond to X events until all windows have been deleted.

There is no automatic evaluation of `.wishrc` in this case, but the script file can always include a reference to the `source` command if desired.

VARIABLES

`Wish` sets the following Tcl variables:

`argc` Contains a count of the number of *arg* arguments (0 if none), not including the options described above.

`argv` Contains a Tcl list whose elements are the *arg* arguments (not including the options described above), in order, or an empty string if there are no *arg* arguments.

`argv0` Contains *fileName* if it was specified. Otherwise, contains the name by which `wish` was invoked.

`tcl_interactive`
Contains 1 if `wish` is reading commands interactively (no **`-file`** option was specified and standard input is a terminal-like device), 0 otherwise.

SCRIPT FILES

If you create a Tcl script in a file whose first line is

```
#!/usr/local/bin/wish -f
```

then you can invoke the script file directly from your shell if you mark it as executable. This assumes that wish has been installed in the default location in `/usr/local/bin`; if it's installed somewhere else then you'll have to modify the above line to match.

PROMPTS

When `wish` is invoked interactively it normally prompts for each command with "`%` ". You can change the prompt by setting the variables `tcl_prompt1` and `tcl_prompt2`. If variable `tcl_prompt1` exists then it must consist of a Tcl script to output a prompt; instead of outputting a prompt `wish` will evaluate the script in `tcl_prompt1`. The variable `tcl_prompt2` is used in a similar way when a newline is typed but the current command isn't yet complete; if `tcl_prompt2` isn't set then no prompt is output for incomplete commands.

KEYWORDS

shell, toolkit

NAME

`xargs` – build and execute command lines from stdin

SYNOPSIS

xargs [-0prtx] [-e[*eof-str***]] [-i[***replace-str***]] [-l[***max-lines***]]**
[-n *max-args***] [-s** *max-chars***] [-P** *max-procs***] [--null]**
[--eof[=*eof-str***] [--replace[=***replace-str***]**
[--max-lines[=*max-lines***] [--interactive]**
[--max-chars max-chars] [--verbose] [--exit]
[--max-procs max-procs] [--no-run-if-empty]
[*command* **[***initial-arguments***]]**

DESCRIPTION

This manual page documents the GNU version of `xargs`. `xargs` reads arguments from the standard input, delimited by blanks (which can be protected with double or single quotes or a backslash) or newlines, and executes the `command` (default is `/bin/echo`) one or more times with any `initial-arguments` followed by arguments read from standard input. Blank lines on the standard input are ignored.

`xargs` exits with the following status:

```
0 if it succeeds
123 if any invocation of the command exited with status 1-
125
124 if the command exited with status 255
125 if the command is killed by a signal
126 if the command cannot be run
127 if the command is not found
1 if some other error occurred.
```

OPTIONS

`--null, -0` Input filenames are terminated by a null character instead of by whitespace and the quotes and backslash are not special (every character is taken literally). However, any end of file string is still recognized. Useful with the GNU `find -print()` option when filenames might contain white space, quote marks, or backslashes.

`--eof[=`*eof-str*`], -e[`*eof-str*`]`
Set the end of file string to *eof-str*. (If the end of file string occurs as a line of input, the rest of the input is ignored.) If *eof-str* is omitted, there is no end of file string. If this option is not given, the end of file string defaults to _.

`--replace[=`*replace-str*`], -i[`*replace-str*`]`
Replace occurrences of *replace-str* in the initial arguments with names read from standard input. If *replace-str* is omitted, it defaults to `{}` (like for `find -exec`). Causes the command to be executed once per line of input. Implies `-x`.

`--max-lines[=`*max-lines*`], -l[`*max-lines*`]`
Use at most *max-lines* nonblank input lines per command line; *max-lines* defaults to 1 if omitted. Trailing blanks cause an input line to be logically continued on the next input line. Implies `-x`.

`--max-args` *max-args*`, -n` *max-args*
Use at most *max-args* arguments per command line. Fewer than *max-args* arguments will be used if the size (see the `-s` option) is exceeded, unless the `-x` option is given, in which case `xargs` will exit.

`--interactive, -p`
Prompt the user about whether to run each command line and read a line from the terminal. Only run the command line if the response starts with `y` or `Y`. Implies `-t`.

`--no-run-if-empty, -r`
If the standard input does not contain any nonblanks, do not run the command. Normally, the command is run once even if there is no input.

`--max-chars` *max-chars*`, -s` *max-chars*
Use at most *max-chars* characters per command line, including the command and initial arguments. The default is as large as possible.

`--verbose, -t` Print the command line on the standard error output before executing it.

`--exit, -x` Exit if the size (see the `-s` option) is exceeded.

`--max-prods` *max-procs*, `-P` *max-procs*
Run up to *max-procs* processes at a time; the default is 1. If *max-procs* is 0, `xargs` will run as many processes as possible at a time. Use the `-n` option with `-P`; otherwise chances are that only one exec will be done.

The long-named options can be introduced with + as well as -, for compatibility with previous releases. Eventually support for + will be removed, because it is incompatible with the POSIX.2 standard.

COPYRIGHT

NAME

`xntpd` – Network Time Protocol daemon

SYNOPSIS

xntpd [-abdm] [-c *conffile* **] [-e** *authdelay* **]**
[-f *driftfile* **] [-k** *keyfile* **] [-l** *logfile* **] [-p** *pidfile* **]**
[-r *broadcastdelay* **] [-s** *statsdir* **] [-t** *trustedkey* **]**
[-v *variable* **] [-V** *variable* **]**

DESCRIPTION

`xntpd` is a daemon which sets and maintains a UNIX system time-of-day in agreement with Internet standard time servers. `xntpd` is a complete implementation of the Network Time Protocol (NTP) version 3 standard, as defined by RFC 1305, but also retains compatibility with version 1 and 2 servers as defined by RFC 1059 and RFC 1119, respectively. `xntpd` does all computations in fixed point arithmetic and requires no floating point code. The computations done in the protocol and clock adjustment code are carried out with high precision and with attention to the details which might introduce systematic bias into the computations, to try to maintain an accuracy suitable for synchronizing with even the most precise external time source.

Ordinarily, `xntpd` reads its configuration from a configuration file at startup time. The default configuration file name is `/etc/ntp.conf`, although this may be overridden from the command line. It is also possible to specify a working, although limited, `xntpd` configuration entirely on the command line, obviating the need for a configuration file. This may be particularly appropriate when `xntpd` is to be configured as a broadcast or multicast client, with all peers being determined by listening to broadcasts at run time. Various internal `xntpd` variables can be displayed and configuration options altered while the daemon is running through use of the `ntpq` and `xntpdc` programs.

The daemon can operate in any of several modes, including symmetric active/passive, client/server and broadcast/multicast. A broadcast/multicast client can automatically discover remote servers, compute one-way delay correction factors and configure itself automatically. This makes it possible to deploy a fleet of workstations without specifying a configuration file or configuration details specific to its environment.

The following command line arguments are understood by `xntpd` (see the configuration file description for a more complete functional description):

-a	run in "authenticate" mode
-b	listen for broadcast NTP and sync to this if available
-c	specify an alternate configuration file
-d	specify debugging mode. This flag may occur multiple times, with each occurrence indicating greater detail of display.
-e	specify the time (in seconds) it takes to compute the NTP encryption field on this computer
-f *driftfile*	specify the location of the drift file
-k	specify the location of the file which contains the NTP authentication keys
-l	logfile specify a log file instead of logging to syslog
-m	listen for multicast messages and synchronize to them if available (requires multicast kernel)
-p	specify the name of the file to record the daemon's process id
-r	ordinarily, the daemon automatically compensates for the network delay between the broadcast/multicast server and the client; if the calibration procedure fails, use the specified the default delay (in seconds)
-s	specify the directory to be used for creating statistics files
-t *trustedkey*	add a key number to the trusted key list
-v	add a system variable
-V	add a system variable listed by default

CONFIGURATION OPTIONS

`xntpd`'s configuration file format is similar to other UNIX configuration files. Comments begin with a "#" character and extend to the end of the line. Blank lines are ignored. Configuration commands consist of an initial keyword followed by a list of arguments, some of which may be optional, separated by whitespace. These commands may not be continued over multiple lines. Arguments may be host names, host addresses written in numeric, dotted-quad form, integers, floating point numbers (when specifying times in seconds) and text strings. Optional arguments are delimited by "[]" in the following descriptions, while alternatives are separated by "|".

```
peer host_address [ key # ] [ version # ] [ prefer ]
server host_address [ key # ] [ version # ] [ prefer ][ mode #]
broadcast host_address [ key # ] [ version # ] [ ttl # ]
```

These three commands specify various time servers to be used and/or time services to be provided. The peer command specifies that the local server is to operate in "symmetric active" mode with the remote server host_address named in the command. In this mode the local server can be synchronized to the remote server and, in addition, the remote server can be synchronized by the local server. This is useful in a network of servers where, depending on various failure scenarios, either the local or remote server host may be the better source of time. The server command specifies that the local server is to operate in "client" mode with the remote server named in the command. In this mode the local server can be synchronized to the remote server, but the remote server can never be synchronized to the local server. The broadcast command specifies that the local server is to operate in "broadcast" mode where the local server sends periodic broadcast messages to a client population at the broadcast/multicast address named in the command. Ordinarily, this specification applies only to the local server operating as a transmitter; for operation as a broadcast client, see the `broadcastclient` or `multicastclient` commands elsewhere in this document. In this mode the host_address is usually the broadcast address on [one of] the local network[s] or a multicast address assigned to NTP. The Numbers Czar has assigned the address 224.0.1.1 to NTP; this is presently the only number that should be used. Note that the use of multicast features requires a multicast kernel, which is not yet ubiquitous in vendor products.

The key option, when included, indicates that all packets sent to the address are to include authentication fields encrypted using the specified key number (the range of which is that of an unsigned 32 bit integer). The default is to not include an encryption field. The version option allows one to specify the version number to be used for outgoing NTP packets. Versions 1, 2, and 3 are the choices, version 3 is the default. The prefer option marks the host as a preferred host. All other things being equal, this host will be chosen for synchronization among a set of correctly operating hosts. The **`ttl`** option is used only with the broadcast mode. It specifies the time-to- live (TTL) to use on multicast packets. Selection of the proper value, which defaults to 127, is something of a black art and must be coordinated with the network admistrator(s).

```
broadcastclient
```

This directs the local server to listen for broadcast messages on the local network, in order to discover other servers on the same subnet. Upon hearing a broadcast message for the first time, the local server measures the nominal network delay using a brief client/server exchange with the remote server, then enters the “broadcastclient” mode, in which it listens for and synchronizes to succeeding broadcast messages. Note that, in order to avoid accidental or malicious disruption in this mode, both the local and remote servers must operate using authentication and the same trusted key and key identifier.

```
multicastclient [ IP address ... ]
```

This command is used in the same way as the `broadcastclient` command, but operates using IP multicasting. Support for this function requires a multicast kernel and the use of authentication. If one or more IP addresses are given, the server joins the respective multicast group(s). If none are given, the IP address assigned to NTP (224.0.1.1) is assumed.

```
driftfile filename
```

This command specifies the name of the file used to record the frequency offset of the local clock oscillator. If the file exists, it is read at startup in order to set the initial frequency offset and then updated once per hour with the current offset computed by the daemon. If the file does not exist or this command is not given, the initial frequency offset is assumed zero. In this case, it may take some hours for the frequency to stabilize and the residual timing errors to subside. The file contains a single floating point value equal to the offset in parts-per-million (ppm). Note that the file is updated by first writing the current drift value into a temporary file and then using rename(3) to replace the old version. This implies that `xntpd` must have write permission for the directory the drift file is located in, and that file system links, symbolic or otherwise, should probably be avoided.

```
enable auth|bclient|pll|monitor|stats [ ... ]
```

Provides a way to enable various server options. Flags not mentioned are unaffected. The **`auth`** flag causes the server to synchronize with unconfigured peers only if the peer has been correctly authenticated using a trusted key and key identifier. The default for this flag is disable (off). The **`bclient`** flag causes the server to listen for a message from a broadcast or multicast server, following which an association is automatically instantiated for that server. The default for this flag is disable (off). The **`pll`** flag enables the server to adjust its local clock, with default enable (on). If not set, the local clock free-runs at its intrinsic time and frequency offset. This flag is useful in case the local clock is controlled by some other device or protocol and NTP is used only to provide synchronization to other clients. The **`monitor`** flag enables the monitoring facility (see elsewhere), with default enable (on). The **`stats`** flag enables statistics facility `filegen` (see description elsewhere.), with default enable (on).

```
disable auth|bclient|pll|monitor|stats [ ... ]
```

Provides a way to disable various server options. Flags not mentioned are unaffected. The flags presently available are described under the enable command.

AUTHENTICATION OPTIONS

```
keys filename
```

This command specifies the name of a file which contains the encryption keys and key identifiers used by `xntpd` when operating in authenticated mode. The format of this file is described later in this document.

```
trustedkey # [ # ... ]
```

This command is used to specify the encryption key identifiers which are trusted for the purposes of authenticating peers suitable for synchronization. The authentication procedures require that both the local and remote servers share the same key and key identifier for this purpose, although different keys can be used with different servers. The arguments are 32 bit unsigned integers. Note, however, that NTP key 0 is fixed and globally known. If meaningful authentication is to be performed the 0 key should not be trusted.

```
requestkey #
```

This command specifies the key identifier to use with the `xntpdc` program, which is useful to diagnose and repair problems that affect `xntpd` operation. The operation of the `xntpdc` program are specific to this particular implementation of `xntpd` and can be expected to work only with this and previous versions of the daemon. Requests from a remote `xntpdc` program which affect the state of the local server must be authenticated, which requires bot the remote program and local server share a common key and key identifier. The argument to this command is a 32 bit unsigned integer. If no `requestkey` command is included in the configuration file, or if the keys don't match, such requests will be ignored.

```
controlkey #
```

This command specifies the key identifier to use with the `ntpq` program, which is useful to diagnose and repair problems that affect `xntpd` operation. The operation of the `ntpq` program and `xntpd` conform to those specified in RFC 1305. Requests from a remote `ntpq` program which affect the state of the local server must be authenticated, which requires bot the remote program and local server share a common key and key identifier. The argument to this command is a 32 bit unsigned integer. If no `requestkey` command is included in the configuration file, or if the keys don't match, such requests will be ignored.

```
authdelay seconds
```

Indicates the amount of time it takes to encrypt an NTP authentication field on the local computer. This value is used to correct transmit timestamps when the authentication is used on outgoing packets. The value usually lies somewhere in the range 0.0001 seconds to 0.003 seconds, though it is very dependent on the CPU speed of the host computer. The value is usually computed using the authspeed program included with the distribution.

ACCESS CONTROL OPTIONS

```
restrict address [ mask numeric_mask ] [ flag ] [ ... ]
```

`xntpd` implements a general purpose address-and-mask based restriction list. The list is sorted by address and by mask, and the list is searched in this order for matches, with the last match found defining the restriction flags associated with the incoming packets. The source address of incoming packets is used for the match, with the 32 bit address being and'ed with the mask associated with the restriction entry and then compared with the entry's address (which has also been and'ed with the mask) to look for a match. The **`mask`** argument defaults to 255.255.255.255, meaning that the **`address`** is treated as the address of an individual host. A default entry (address 0.0.0.0, mask 0.0.0.0) is always included and, given the sort algorithm, is always the first entry in the list. Note that, while **`address`** is normally given in dotted-quad format, the text string **`default`**, with no mask option, may be used to indicate the default entry.

In the current implementation, flags always restrict access, i.e. an entry with no flags indicates that free access to the server is to be given. The flags are not orthogonal, in that more restrictive flags will often make less restrictive ones redundant. The flags can generally be classed into two categories, those which restrict time service and those which restrict informational queries and attempts to do run time reconfiguration of the server. One or more of the following flags may be specified:

`ignore` Ignore all packets from hosts which match this entry. If this flag is specified neither queries nor time server polls will be responded to.

`noquery` Ignore all NTP mode 6 and 7 packets (i.e. information queries and configuration requests) from the source. Time service is not affected.

`nomodify` Ignore all NTP mode 6 and 7 packets which attempt to modify the state of the server (i.e. run time reconfiguration). Queries which return information are permitted.

notrap Decline to provide mode 6 control message trap service to matching hosts. The trap service is a subsystem of the mode 6 control message protocol which is intended for use by remote event logging programs.

lowpriotrap Declare traps set by matching hosts to be low priority. The number of traps a server can maintain is limited (the current limit is 3). Traps are usually assigned on a first come, first served basis, with later trap requestors being denied service. This flag modifies the assignment algorithm by allowing low priority traps to be overridden by later requests for normal priority traps.

noserve Ignore NTP packets whose mode is other than 6 or 7. In effect, time service is denied, though queries may still be permitted.

nopeer Provide stateless time service to polling hosts, but do not allocate peer memory resources to these hosts even if they otherwise might be considered useful as future synchronization partners.

notrust Treat these hosts normally in other respects, but never use them as synchronization sources.

limited These hosts are subject to limitation of number of clients from the same net. Net in this context refers to the IP notion of net (class A, class B, class C, etc.). Only the first **client_limit** hosts that have shown up at the server and that have been active during the last **client_limit_period** seconds are accepted. Requests from other clients from the same net are rejected. Only time request packets are taken into account. “Private”, “control”, and “broadcast” packets are not subject to client limitation and therefore are not contributing to client count. History of clients is kept using the monitoring capability of `xntpd`. Thus, monitoring is active as long as there is a restriction entry with the **limited** flag. The default value for **client_limit** is 3. The default value for **client_limit_period** is 3600 seconds.

ntpport This is actually a match algorithm modifier, rather than a restriction flag. Its presence causes the restriction entry to be matched only if the source port in the packet is the standard NTP UDP port (123). Both **ntpport** and non-**ntpport** may be specified. The **ntpport** is considered more specific and is sorted later in the list.

Default restriction list entries, with the flags “ignore, ntpport”, for each of the local host’s interface addresses are inserted into the table at startup to prevent the server from attempting to synchronize to its own time. A default entry is also always present, though if it is otherwise unconfigured no flags are associated with the default entry (i.e. everything besides your own NTP server is unrestricted).

The restriction facility was added to allow the current access policies of the time servers running on the NSFnet backbone to be implemented with `xntpd` as well. While this facility may be otherwise useful for keeping unwanted or broken remote time servers from affecting your own, it should not be considered an alternative to the standard NTP authentication facility. Source address based restrictions are easily circumvented by a determined cracker.

```
clientlimit limit
```

Sets client_limit to limit, allows configuration of client limitation policy. This variable defines the number of clients from the same network that are allowed to use the server.

```
clientperiod period
```

Sets client_limit_period, allows configuration of client limitation policy. This variable specifies the number of seconds after which a client is considered inactive and thus no longer is counted for client limit restriction.

MONITORING OPTIONS

```
statsdir /directory path/
```

Indicates the full path of a directory where statistics files should be created (see below). This keyword allows the (otherwise constant) filegen filename prefix to be modified for file generation sets used for handling statistics logs (see filegen statement below).

```
statistics name...
```

Enables writing of statistics records. Currently, three kinds of statistics are supported:

- loopstats
- peerstats
- clockstats

`loopstats` enables recording of loop filter statistics information. Each update of the local clock outputs a line of the following form to the file generation set named “loopstats”:

```
48773 10847.650 0.0001307 17.3478 2
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next three fields show time offset in seconds, frequency offset in parts-per-million and time constant of the clock-discipline algorithm at each update of the clock.

`peerstats` enables recording of peer statistics information. This includes statistics records of all peers of a NTP server and of the 1-pps signal, where present and configured. Each valid update appends a line of the following form to the current element of a file generation set named “peerstats”:

```
48773 10847.650 127.127.4.1 9714 -0.001605 0.00000 0.00142
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next two fields show the peer address in dotted-quad notation and status, respectively. The status field is encoded in hex in the format described in Appendix A of the NTP specification RFC 1305. The final three fields show the offset, delay and dispersion, all in seconds.

`clockstats` enables recording of clock driver statistics information. Each update received from a clock driver outputs a line of the following form to the file generation set named "clockstats":

```
49213 525.624 127.127.4.1 93 226 00:08:29.606 D
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next field shows the clock address in dotted-quad notation, The final field shows the last timecode received from the clock in decoded ASCII format, where meaningful. In some clock drivers a good deal of additional information can be gathered and displayed as well. See information specific to each clock for further details.

Statistic files are managed using file generation sets (see `filegen` below). The information obtained by enabling statistics recording allows analysis of temporal properties of a `xntpd` server. It is usually only useful to primary servers or maybe main campus servers.

```
filegen name [ file filename ] [ type typename ]
  [ flag flagval ] [ linknolink ] [ enabledisable ]
```

Configures setting of generation file set name. Generation file sets provide a means for handling files that are continuously growing during the lifetime of a server. Server statistics are a typical example for such files. Generation file sets provide access to a set of files used to store the actual data. At any time at most one element of the set is being written to. The type given specifies when and how data will be directed to a new element of the set. This way, information stored in elements of a file set that are currently unused are available for administration operations without the risk of disturbing the operation of `xntpd`. (Most important: they can be removed to free space for new data produced.) File names of set members are built from three elements.

`prefix` This is a constant file name path. It is not subject to modifications via the `filegen` statement. It is defined by the server, usually specified as a compile time constant. It may, however, be configurable for individual file generation sets via other commands. For example, the prefix used with “loopstats” and “peerstats” filegens can be configured using the statsdir statement explained above.

filename This string is directly concatenated to the prefix mentioned above (no intervening'/' (slash)). This can be modified using the “file” argument to the “filegen” statement. No “..” elements are allowed in this component to prevent file names referring to parts outside the file system hierarchy denoted by “prefix”.

suffix This part is reflects individual elements of a file set. It is generated according to the type of a file set as explained below.

A file generation set is characterized by its type. The following types are supported:

none The file set is actually a single plain file.

pid One element of file set is used per incarnation of a `xntpd` server. This type does not perform any changes to file set members during runtime, however it provides an easy way of separating files belonging to different `xntpd` server incarnations. The set member file name is built by appending a dot ('.') to concatenated “prefix” and “filename” strings, and appending the decimal representation of the process id of the `xntpd` server process.

day One file generation set element is created per day. The term day is based on UTC. A day is defined as the period between 00:00 and 24:00 UTC. The file set member suffix consists of a dot “.” and a day specification in the form <YYYYMMDD>. YYYY is a 4 digit year number (e.g. 1992). MM is a two digit month number. DD is a two digit day number. Thus, all information written at December 10th, 1992 would end up in a file named “<prefix><filename>.19921210”.

week Any file set member contains data related to a certain week of a year. The term week is defined by computing "day of year" modulo 7. Elements of such a file generation set are distinguished by appending the following suffix to the file set file name base: A dot, a four digit year number, the letter "W", and a two digit week number. For example, information from January, 10th 1992 would end up in a file with suffix ".1992W1".

month One generation file set element is generated per month. The file name suffix consists of a dot, a four digit year number, and a two digit month.

year One generation file element is generated per year. The file name suffix consists of a dot and a 4 digit year number.

age This type of file generation sets changes to a new element of the file set every 24 hours of server operation. The file name suffix consists of a dot, the letter "a", and an eight digit number. This number is taken to be the number of seconds the server is running at the start of the corresponding 24 hour period.

Information is only written to a file generation set when this set is "enabled". Output is prevented by specifying "disabled".

It is convenient to be able to access the current element of a file generation set by a fixed name. This feature is enabled by specifying "link" and disabled using "nolink". If "link" is specified, a hard link from the current file set element to a file without suffix is created. When there is already a file with this name and the number of links of this file is one, it is renamed appending a dot, the letter "C", and the pid of the `xntpd` server process. When the number of links is greater than one, the file is unlinked. This allows the current file to be accessed by a constant name.

MISCELLANEOUS OPTIONS

```
precision #
```

This command specifies the nominal precision of the local clock. The value is an integer approximately equal to the base 2 logarithm of the local timekeeping precision in seconds. Normally, the daemon determines the precision automatically at startup, so this command is necessary only in special cases when the precision cannot be determined automatically.

```
broadcastdelay seconds
```

The broadcast and multicast modes require a special calibration to determine the network delay between the local and remote servers. Ordinarily, this is done automatically by the initial protocol exchanges between the local and remote servers. In some cases, the calibration procedure may fail due to network or server access controls, for example. This command specifies the default delay to be used under these circumstances. Typically (for Ethernet), a number between 0.003 and 0.007 seconds is appropriate. The default when this command is not used is 0.004 seconds.

```
trap host_address [ port port_number ] [ interface
interface_addess ]
```

This command configures a trap receiver at the given host address and port number for sending messages with the specified local interface address. If the port number is unspecified a value of 18447 is used. If the interface address is not specified the message is sent with a source address which is that of the local interface the message is sent through. Note that on a multihomed host the interface used may vary from time to time with routing changes.

The trap receiver will generally log event messages and other information from the server in a log file. While such monitor programs may also request their own trap dynamically, configuring a trap receiver will ensure that no messages are lost when the server is started.

```
setvar variable [default]
```

This command adds an additional system variable. These variables can be used to distribute additional information such as the access policy. If the variable of the from `<name>=<value>` is followed by the default keyword the variable will be listed as part of the default system variables (`ntpq rv` command). These additional variables serve informational purposes only. They are not related to the protocol other that they can be listed. The known protocol variables will always override any variables defined via the setvar mechanism.

There are three special variables that contain the names of all variable of the same group. The `sys_var_list` holds the names of all system variables. The `peer_var_list` holds the names of all peer variables and the `clock_var_list` hold the names of the reference clock variables.

```
monitor yes|no authenticate yes|no
```

These commands have been superseded by the enable and disable commands. They are listed here for historical purposes.

AUTHENTICATION KEY FILE FORMAT

The NTP standard specifies an extension allowing verification of the authenticity of received NTP packets, and to provide an indication of authenticity in outgoing packets. This is implemented in `xntpd` using the DES or MD5 algorithms to compute a digital signature, or message-digest. The specification allows any one of possibly 4 billion keys, numbered with 32 bit key identifiers, to be used to authenticate an association. The servers involved in an association must agree on the key and key identifier used to authenticate their data, though they must each learn the key and key identifier independently. In the case of DES, the keys are 56 bits long with, depending on type, a parity check on each byte. In the case of MD5, the keys are 64 bits (8 bytes). `xntpd` reads its keys from a file specified using the -k command line option or the keys statement in the configuration file. While key number 0 is fixed by the NTP standard (as 56 zero bits) and may not be changed, one or more of the keys numbered 1 through 15 may be arbitrarily set in the keys file.

The key file uses the same comment conventions as the configuration file. Key entries use a fixed format of the form

```
keyno type key
```

where “keyno” is a positive integer, “type” is a single character which defines the format the key is given in, and “key” is the key itself.

The key may be given in one of three different formats, controlled by the “type” character. The three key types, and corresponding formats, are listed following.

S The “key” is a 64 bit hexadecimal number in the format specified in the DES document, that is the high order 7 bits of each octet are used to form the 56 bit key while the low order bit of each octet is given a value such that odd parity is maintained for the octet. Leading zeroes must be specified (i.e. the key must be exactly 16 hex digits long) and odd parity must be maintained. Hence a zero key, in standard format, would be given as 0101010101010101.

N The “key” is a 64 bit hexadecimal number in the format specified in the NTP standard. This is the same as the DES format except the bits in each octet have been rotated one bit right so that the parity bit is now the high order bit of the octet. Leading zeroes must be specified and odd parity must be maintained. A zero key in NTP format would be specified as 8080808080808080

A The “key” is a 1-to-8 character ASCII string. A key is formed from this by using the lower order 7 bits of the ASCII representation of each character in the string, with zeroes being added on the right when necessary to form a full width 56 bit key, in the same way that encryption keys are formed from UNIX passwords.

M The “key” is a 1-to-8 character ASCII string, using the MD5 authentication scheme. Note that both the keys and the authentication schemes (DES or MD5) must be identical between a set of peers sharing the same key number.

One of the keys may be chosen, by way of the configuration file requestkey statement, to authenticate run time configuration requests made using the `xntpdc` program. The latter program obtains the key from the terminal as a password, so it is generally appropriate to specify the key chosen to be used for this purpose in ASCII format.

PRIMARY CLOCK SUPPORT

`xntpd` can be optionally compiled to include support for a number of types of reference clocks. A reference clock will generally (though not always) be a radio timecode receiver which is synchronized to a source of standard time such as the services offered by the NRC in Canada and NIST in the U.S. The interface between the computer and the timecode receiver is device dependent and will vary, but is often a serial port.

Support for the various reference clock drivers is conditionally compiled using the compiler define codes described elsewhere. An attempt to configure a reference clock when specific support is not available or the hardware port has not been appropriately configured results in a scolding remark to the system log file, but is otherwise non hazardous.

For the purposes of configuration, `xntpd` treats reference clocks in a manner analogous to normal NTP peers as much as possible. Reference clocks are referred to by address, much as a normal peer is, though an invalid IP address is used to distinguish them from normal peers. Reference clock addresses are of the form 127.127.t.u where t is an integer denoting the clock type and u indicates the type-specific unit number. Reference clocks are configured using a server statement in the configuration file where the `host_address` is the clock address. The key, version and **`ttl`** options are not used for reference clock support. Some reference clocks require a mode option to further specify their operation. The permissible values for reference clock types, units and any applicable modes are documented in `README.refclock` in the source directory. The prefer option can be useful to persuade the server to cherish a reference clock with somewhat more enthusiasm than other reference clocks or peers, if this is advisable. Clock addresses may generally be used anywhere in the configuration file a normal IP address can be used, for example, in restrict statements, although such use would normally be considered strange.

Reference clock support provides the fudge command, which can be used to configure reference clocks in special ways. Following is the generic format that applies to this command

```
fudge 127.127.t.u [ time1 secs ] [ time2 secs ]
  [ stratum int ] [ refid int ] [ flag1 0|1 ] [ flag2 0|1 ]
  [ flag3 0|1 ] [ flag4 0|1 ]
```

The time1 and time2 options are specified in fixed point seconds and used in some clock drivers as calibration constants. By convention, and unless indicated otherwise, time1 is used as a calibration constant to adjust the nominal time offset of a particular clock to agree with an external standard, such as a precision PPS signal. The specified offset is in addition to the propagation delay provided by other means, such as internal DIPswitches. The stratum option is a number in the range zero to 15 and is used to assign a non-standard operating stratum to the clock. The **refid** option is an ASCII string in the range one to four characters and is used to assign a nonstandard reference identifier to the clock. Finally, the four binary flags flag1, flag2, flag3 and flag4 are used for customizing the clock driver. The interpretation of these values, and whether they are used at all, is a function of the needs of the particular clock driver. However, by convention, and unless indicated otherwise, flag3 is used to attach the ppsclock streams module to the configured driver, while flag4 is used to enable recording verbose monitoring data to the clockstats file configured with the `filegen` command. Further information on the ppsclock streams module is in the README file in the `./kernel` directory in the current xntp3 program distribution. Further information on this feature is available in the

Ordinarily, the stratum of a reference clock is by default zero. Since the `xntpd` daemon adds one to the stratum of each peer, a primary server ordinarily displays stratum one. In order to provide engineered backups, it is often useful to specify the reference clock stratum as greater than zero. The stratum option is used for this purpose. Also, in cases involving both a reference clock and a 1-pps discipline signal, it is useful to specify the reference clock identifier as other than the default, depending on the driver. The **refid** option is used for this purpose. Except where noted, these options apply to all clock drivers.

xntpd on UNIX machines currently supports several different types of clock hardware plus a special pseudo-clock used for backup or when no other clock source is available. In the case of most of the clock drivers, support for a 1-pps precision timing signal is available as described in the README file in the ./doc directory of the xntp3 program distribution. The clock drivers, and the addresses used to configure them, are described in the README.refclocks in the doc directory of the current program distribution.

VARIABLES

Most variables used by the NTP protocol can be examined with the xntpdc (mode 7 messages) and the ntpq (mode 6 messages). Currently very few variables can be modified via mode 6 messages. These variables are either created with the setvar directive or the leap warning variables. The leap warning bits that can be set in the leapwarning variable (up to one month ahead). Both, the *leapwarning* and in the *leapindication* variable, have a slightly different encoding than the usual leap bits interpretation:

00	The daemon passes the leap bits of its synchronization source (usual mode of operation).
01/10	A leap second is added/deleted (operator forced leap second).
11	Leap information from the astronavigation source is ignored (thus LEAP_NOWARNING is passed on).

FILES

etc/ntp.conf	default name for configuration file
/etc/ntp.drift	conventional name for drift file
/etc/ntp.keys	conventional name for key file

SEE ALSO

Utility Programs- xntpdc, ntpq, ntpdate

HISTORY

Written by Dennis Ferguson at the University of Toronto. Text amended by David Mills at the University of Delaware.

BUGS

`xntpd` has gotten rather fat. While not huge, it has gotten larger than might be desirable for an elevated-priority daemon running on a workstation, particularly since many of the fancy features which consume the space were designed more with a busy primary server, rather than a high stratum workstation, in mind.

COPYRIGHT NOTICE

NAME

`xntpdc` – query/control program for Network Time Protocol daemon

SYNOPSIS

xntpdc [-ilnps] [-c command] [host] [...]

DESCRIPTION

`xntpdc` is used to query the `xntpd` daemon about its current state and to request changes in that state. The program may be run either in interactive mode or controlled using command line arguments. Extensive state and statistics information is available through the `xntpdc` interface. In addition, nearly all the configuration options which can be specified at start up using `xntpd`'s configuration file may also be specified at run time using `xntpdc`.

If one or more request options is included on the command line when `xntpdc` is executed, each of the requests will be sent to the NTP servers running on each of the hosts given as command line arguments, or on localhost by default. If no request options are given, `xntpdc` will attempt to read commands from the standard input and execute these on the NTP server running on the first host given on the command line, again defaulting to localhost when no other host is specified. `xntpdc` will prompt for commands if the standard input is a terminal device.

`xntpdc` uses NTP mode 7 packets to communicate with the NTP server, and hence can be used to query any compatible server on the network which permits it. Note that since NTP is a UDP protocol this communication will be somewhat unreliable, especially over large distances in terms of network topology. `xntpdc` makes no attempt to retransmit requests, and will time requests out if the remote host is not heard from within a suitable time out time.

Command line options are described following. Specifying a command line option other than **-i** or **-n** will cause the specified query (queries) to be sent to the indicated host(s) immediately. Otherwise, `xntpdc` will attempt to read interactive format commands from the standard input.

-c The following argument is interpreted as an interactive format command and is added to the list of commands to be executed on the specified host(s). Multiple **-c** options may be given.

-i Force `xntpdc` to operate in interactive mode. Prompts will be written to the standard output and commands read from the standard input.

-l Obtain a list of peers which are known to the server(s). This switch is equivalent to "-c listpeers".

-n Output all host addresses in dotted-quad numeric format rather than converting to the canonical host names.

-p Print a list of the peers known to the server as well as a summary of their state. This is equivalent to "-c peers".

-s Print a list of the peers known to the server as well as a summary of their state, but in a slightly different format than the **-p** switch. This is equivalent to "-c dmpeers".

INTERNAL COMMANDS

Interactive format commands consist of a keyword followed by zero to four arguments. Only enough characters of the full keyword to uniquely identify the command need be typed. The output of a command is normally sent to the standard output, but optionally the output of individual commands may be sent to a file by appending a ">", followed by a file name, to the command line.

A number of interactive format commands are executed entirely within the `xntpdc` program itself and do not result in NTP mode 7 requests being sent to a server. These are described following.

```
? [ command_keyword }
```

A “?” by itself will print a list of all the command keywords known to this incarnation of `xntpdc`. A “?” followed by a command keyword will print function and usage information about the command. This command is probably a better source of information about `xntpdc` than this manual page.

```
help [ command_keyword ]
```

A synonym for the ? command.

```
timeout millseconds
```

Specify a time out period for responses to server queries. The default is about 8000 milliseconds.

```
delay milliseconds
```

Specify a time interval to be added to timestamps included in requests which require authentication. This is used to enable (unreliable) server reconfiguration over long delay network paths or between machines whose clocks are unsynchronized.

```
host hostname
```

Set the host to which future queries will be sent. Hostname may be either a host name or a numeric (dotted quad) dmaddress.

```
keyid #
```

This command allows the specification of a key number to be used to authenticate configuration requests. This must correspond to the key number the server has been configured to use for this purpose.

```
passwd
```

This command prompts you to type in a password (which will not be echoed) which will be used to authenticate configuration requests. The password must correspond to the key configure for use by the NTP server for this purpose if such requests are to be successful.

```
hostnames yes|no
```

If “yes” is specified, host names are printed in information displays. If “no” is given, numeric addresses are printed instead. The default is “yes” unless modified using the command line -n switch.

```
quit
```

Exit `xntpdc`.

QUERY COMMANDS

Query commands result in NTP mode 7 packets containing requests for information being sent to the server. These are “read-only” commands in that they make no modification of the server configuration state.

```
listpeers
```

Obtains and prints a brief list of the peers for which the server is maintaining state. These should include all configured peer associations as well as those peers whose stratum is such that they are considered by the server to be possible future synchronization candidates.

```
peers
```

Obtains a list of peers for which the server is maintaining state, along with a summary of that state. Summary information includes the address of the remote peer, the local interface address (0.0.0.0 if a local address has yet to be determined), the stratum of the remote peer (a stratum of 16 indicates the remote peer is unsynchronized), the polling interval, in seconds, the reachability register, in octal, and the current estimated delay, offset and dispersion of the peer, all in seconds. In addition, the character in the left margin indicates the mode this peer entry is operating in. A “+” denotes symmetric active, a “-” indicates symmetric passive, a “=” means the remote server is being polled in client mode, a “^” indicates that the server is broadcasting to this address, a “~” denotes that the remote peer is sending broadcasts and a “*” marks the peer the server is currently synchronizing to.

The contents of the host field may be one of four forms. It may be a host name, an IP address, a reference clock implementation name with its parameter or `REFCLK(<implementation number>,<parameter>)`. On “hostnames no” only IP-addresses will be displayed.

```
dmpeers
```

A slightly different peer summary list. Identical to the output of the peers command except for the character in the leftmost column. Characters only appear beside peers which were included in the final stage of the clock selection algorithm. A “.” indicates that this peer was cast off in the falseticker detection, while a “+” indicates that the peer made it through. A “*” denotes the peer the server is currently synchronizing with.

```
showpeer peer_address [ addr2 ] [ addr3 ] [ addr4 ]
```

Shows a detailed display of the current peer variables for one or more peers. Most of these values are described in the NTP Version 2 specification.

```
pstats peer_address [ addr2 ] [ addr3 ] [ addr4 ]
```

Show per-peer statistic counters associated with the specified peer(s).

```
clockinfo clock_peer_address [ addr2 ] [ addr3 ] [ addr4 ]
```

Obtain and print information concerning a peer clock. The values obtained provide information on the setting of fudge factors and other clock performance information.

```
kerninfo
```

Obtain and print kernel phase-lock loop operating parameters. This information is available only if the kernel has been specially modified for a precision timekeeping function.

```
loopinfo [ oneline|multiline ]
```

Print the values of selected loop filter variables. The loop filter is the part of NTP which deals with adjusting the local system clock. The “offset” is the last offset given to the loop filter by the packet processing code. The “frequency” is the frequency error of the local clock in parts- per-million (ppm). The “time_const” controls the “stiffness” of the phase-lock loop and thus the speed at which it can adapt to oscillator drift. The “watchdog timer” value is the number of seconds which have elapsed since the last sample offset was given to the loop filter. The “oneline” and “multiline” options specify the format in which this information is to be printed, with “multiline” as the default.

```
sysinfo
```

Print a variety of system state variables, i.e. state related to the local server. All except the last four lines are described in the NTP Version 3 specification, RFC 1305. The “system flags” show various system flags, some of which can be set and cleared by the “enable” and “disable” configuration commands, respectively. The “stability” is the residual frequency error remaining after the system frequency correction is applied and is intended for maintenance and debugging. In most architectures, this value will initially decrease from as high as 500 ppm to a nominal value in the range .01 to 0.1 ppm. If it remains high for some time after starting the daemon, something may be wrong with the local clock, or the value of the kernel variable “tick” may be incorrect. The “broadcastdelay” shows the default broadcast delay, as set by the "broadcastdelay" configuration command, while the "authdelay" shows the default authentication delay, as set by the "authdelay" configuration command.

```
sysstats
```

Print statistics counters maintained in the protocol module.

```
memstats
```

Print statistics counters related to memory allocation code.

```
iostats
```

Print statistics counters maintained in the input-output module.

```
timerstats
```

Print statistics counters maintained in the timer/event queue support code.

```
reslist
```

Obtain and print the server's restriction list. This list is (usually) printed in sorted order and may help to understand how the restrictions are applied.

```
monlist [ version ]
```

Obtain and print traffic counts collected and maintained by the monitor facility. The version number should not normally need to be specified.

```
clkbug clock_peer_address [ addr2 ] [ addr3 ] [ addr4 ]
```

Obtain debugging information for a reference clock driver. This information is provided only by some clock drivers and is mostly undecodable without a copy of the driver source in hand.

RUNTIME CONFIGURATION REQUESTS

All requests which cause state changes in the server are authenticated by the server using a configured NTP key (the facility can also be disabled by the server by not configuring a key). The key number and the corresponding key must also be made known to `xtnpdc`. This can be done using the `keyid` and `passwd` commands, the latter of which will prompt at the terminal for a password to use as the encryption key. You will also be prompted automatically for both the key number and password the first time a command which would result in an authenticated request to the server is given. Authentication not only provides verification that the requester has permission to make such changes, but also gives an extra degree of protection again transmission errors.

Authenticated requests always include a timestamp in the packet data, which is included in the computation of the authentication code. This timestamp is compared by the server to its receive time stamp. If they differ by more than a small amount the request is rejected. This is done for two reasons. First, it makes simple replay attacks on the server, by someone who might be able to overhear traffic on your LAN, much more difficult. Second, it makes it more difficult to request configuration changes to your server from topologically remote hosts. While the reconfiguration facility will work well with a server on the local host, and may work adequately between time-synchronized hosts on the same LAN, it will work very poorly for more distant hosts. As such, if reasonable passwords are chosen, care is taken in the distribution and protection of keys and appropriate source address restrictions are applied, the run time reconfiguration facility should provide an adequate level of security.

The following commands all make authenticated requests.

```
addpeer peer_address [ keyid ] [ version# ] [ prefer ]
```

Add a configured peer association at the given address and operating in symmetric active mode. Note that an existing association with the same peer may be deleted when this command is executed, or may simply be converted to conform to the new configuration, as appropriate. If the optional **`keyid`** is a nonzero integer, all outgoing packets to the remote server will have an authentication field attached encrypted with this key. If the value is 0 (or not given) no authentication will be done. The **`version#`** can be 1, 2 or 3 and defaults to 3. The **`prefer`** keyword indicates a preferred peer (and thus will be used primarily for clock synchronisation if possible). The preferred peer also determines the validity of the PPS signal—if the preferred peer is suitable for synchronisation so is the PPS signal.

```
addserver peer_address [ keyid ] [ version# ] [ prefer ]
```

Identical to the `addpeer` command, except that the operating mode is client.

```
broadcast peer_address [ keyid ] [ version# ]
```

Identical to the `addpeer` command, except that the operating mode is broadcast. In this case a valid key identifier and key are required. The peer_address parameter can be the broadcast address of the local network or a multicast group address assigned to NTP. If a multicast address, a multicast-capable kernel is required.

```
unconfig peer_address [ addr2 ] [ addr3 ] [ addr4 ]
```

This command causes the configured bit to be removed from the specified peer(s). In many cases this will cause the peer association to be deleted. When appropriate, however, the association may persist in an unconfigured mode if the remote peer is willing to continue on in this fashion.

```
fudge peer_address [ time1 ] [ time2 ] [ stratum ] [ refid ]
```

This command provides a way to set certain data for a reference clock. See the source listing for further information.

```
enable auth|bclient|pll|monitor|stats [ ... ]
```

Provides a way to enable various server options. Flags not mentioned are unaffected. The **auth** flag causes the server to synchronize with unconfigured peers only if the peer has been correctly authenticated using a trusted key and key identifier. The default for this flag is disable (off). The **bclient** flag causes the server to listen for a message from a broadcast or multicast server, following which an association is automatically instantiated for that server. The default for this flag is disable (off). The **pll** flag enables the server to adjust its local clock, with default enable (on). If not set, the local clock free-runs at its intrinsic time and frequency offset. This flag is useful in case the local clock is controlled by some other device or protocol and NTP is used only to provide synchronization to other clients. The **monitor** flag enables the monitoring facility (see elsewhere), with default disable (off). The **stats** flag enables statistics facility `filegen` (see description elsewhere.), with default enable (on).

```
disable auth|bclient|pll|monitor|stats [ ... ]
```

Provides a way to disable various server options. Flags not mentioned are unaffected. The flags presently available are described under the enable command.

```
restrict address mask flag [ flag ]
```

Causes flag(s) to be added to an existing restrict list entry, or adds a new entry to the list with the specified flag(s). The possible choices for the flags arguments are given in the following list:

ignore Ignore all packets from hosts which match this entry. If this flag is specified neither queries nor time server polls will be responded to.

noquery Ignore all NTP mode 7 packets (i.e. information queries and configuration requests) from the source. Time service is not affected.

nomodify Ignore all NTP mode 7 packets which attempt to modify the state of the server (i.e. run time reconfiguration). Queries which return information are permitted.

notrap Decline to provide mode 6 control message trap service to matching hosts. The trap service is a subsystem of the mode 6 control message protocol which is intended for use by remote event logging programs.

lowpriotrap Declare traps set by matching hosts to be low priority. The number of traps a server can maintain is limited (the current limit is 3). Traps are usually assigned on a first come, first served basis, with later trap requestors being denied service. This flag modifies the assignment algorithm by allowing low priority traps to be overridden by later requests for normal priority traps.

noserve	Ignore NTP packets whose mode is other than 7. In effect, time service is denied, though queries may still be permitted.
nopeer	Provide stateless time service to polling hosts, but do not allocate peer memory resources to these hosts even if they otherwise might be considered useful as future synchronization partners.
notrust	Treat these hosts normally in other respects, but never use them as synchronization sources.
limited	These hosts are subject to limitation of number of clients from the same net. Net in this context refers to the IP notion of net (class A, class B, class C, etc.). Only the first **client_limit** hosts that have shown up at the server and that have been active during the last **client_limit_period** seconds are accepted. Requests from other clients from the same net are rejected. Only time request packets are taken into account. **Private**, **control**, and **broadcast** packets are not subject to client limitation and therefore are not contributing to client count. History of clients is kept using the monitoring capability of `xntpd`. Thus, monitoring is active as long as there is a restriction entry with the **limited** flag. The default value for **client_limit** is 3. The default value for **client_limit_period** is 3600 seconds. Currently both variables are not runtime configurable.
ntpport	This is actually a match algorithm modifier, rather than a restriction flag. Its presence causes the restriction entry to be matched only if the source port in the packet is the standard NTP UDP port (123). Both **ntpport** and non-**ntpport** may be specified. The **ntpport** is considered more specific and is sorted later in the list.

```
unrestrict address mask flag [ flag ]
```

Remove the specified flag(s) from the restrict list entry indicated by the address and mask arguments.

```
delrestrict address mask [ ntpport ]
```

Delete the matching entry from the restrict list.

```
monitor yes|no
```

Enable or disable the monitoring facility. Note that a monitor no command followed by a monitor yes command is a good way of resetting the packet counts.

```
readkeys
```

Causes the current set of authentication keys to be purged and a new set to be obtained by rereading the keys file (which must have been specified in the `xntpd` configuration file). This allows encryption keys to be changed without restarting the server.

```
trustkey keyid [ keyid ] [ keyid ] [ keyid ]
```

Adds one or more keys to the trusted key list. When authentication is enabled, peers whose time is to be trusted must be authenticated using a trusted key.

```
untrustkey keyid [ keyid ] [ keyid ] [ keyid ]
```

Removes one or more keys from the trusted key list.

```
authinfo
```

Returns information concerning the authentication module, including known keys and counts of encryptions and decryptions which have been done.

```
setprecision precision_value
```

Sets the precision which the server advertises to the specified value. This should be a negative integer in the range -4 through -20.

```
traps
```

Display the traps set in the server. See the source listing for further information.

```
addtrap address [ port ] [ interface ]
```

Set a trap for asynchronous messages. See the source listing for further information.

```
clrtrap address [ port ] [ interface ]
```

Clear a trap for asynchronous messages. See the source listing for further information.

```
reset ...
```

Clear the statistics counters in various modules of the server. See the source listing for further information.

SEE ALSO

Utility Programs- `xntpd`

HISTORY

Written by Dennis Ferguson at the University of Toronto.

COPYRIGHT NOTICE

NAME

`zdiff`, `zcmp` – compare compressed files

SYNOPSIS

`zcmp` **[***cmp_options* **]** *file1* **[** *file2* **]**

`zdiff` **[***diff_options* **]** *file1* **[** *file2* **]**

DESCRIPTION

`zdiff` and `zcmp` are used to invoke the `cmp` or the `diff` program on compressed files. All options specified are passed directly to `cmp` or `diff`. If only one file is specified, then the files compared are *file1* and an uncompressed *file1*.gz. If two files are specified, then they are uncompressed if necessary and fed to `cmp` or `diff`.

The exit status from `cmp` or `diff` is preserved.

SEE ALSO

Utility Programs - `cmp, diff, zmore, zgrep, znew, zforce, gzip, gzexe`

BUGS

Messages from the `cmp` or `diff` programs refer to temporary filenames instead of those specified.

NAME

`zeronode` – re-initialize a disk inode

SYNOPSIS

`zeronode` *device inum ...*

`zeronode` *file*

DESCRIPTION

`zeronode` re-initializes certain fields in the specified disk inode(s), and should only be used as a last resort when attempting to fix an already-corrupted file system. In the first invocation, `zeronode` re-initializes each disk inode *inum* on the block *device*. In the second invocation, `zeronode` re-initializes the disk inode associated with *file*.

The following fields in the `d_inode` structure, defined in `disk.h`, are modified by `zeronode`:

`i_mode`	Set equal to `S_IFREG	0666` (regular file with read/write permissions).
`i_nlink`	Set equal to 1.
`i_genid`	Set equal to 0.
`i_size`	Set equal to 0.
`i_blksize`	Set equal to 0.
`i_blocks[]`	Each entry in the array is set equal to 0.

All other fields are unmodified by `zeronode`.

SEE ALSO

Utility Programs - `shownode`

System Calls - `stat()`

NAME

`zforce` – force a `.gz` extension on all gzip files

SYNOPSIS

`zforce [` *name ...* **`]`**

DESCRIPTION

`zforce` forces a `.gz` extension on all `gzip` files so that `gzip` will not compress them twice. This can be useful for files with names truncated after a file transfer. On systems with a 14-char limitation on file names, the original name is truncated to make room for the `.gz` suffix. For example, `12345678901234` is renamed to `12345678901.gz`. A file name such as `foo.tgz` is left intact.

SEE ALSO

Utility Programs - `gzip, znew, zmore, zgrep, zdiff, gzexe`

NAME

zgrep – search (possibly compressed) files

SYNOPSIS

zgrep [*grep_options***] [-r]** *pattern filename ...*

DESCRIPTION

zgrep is used to invoke the grep on compress'ed or gzip'ed files. All options specified are passed directly to grep. If no file is specified, then the standard input is decompressed if necessary and fed to grep. Otherwise, the given files are uncompressed if necessary and fed to grep.

If zgrep is invoked as zegrep or zfgrep then egrep or fgrep is used instead of grep. If the **GREP** environment variable is set, zgrep uses it as the grep program to be invoked. For example:

for sh: GREP=fgrep zgrep *string files*

for csh: (setenv GREP fgrep; zgrep *string files*)

AUTHOR

Charles Levert (*charles@comm.polymtl.ca*)

SEE ALSO

Utility Programs - grep, egrep, fgrep, zdiff, zmore, znew, zforce, gzip, gzexe

NAME

`zmore` – filter for CRT viewing of compressed text

SYNOPSIS

`zmore [` *name ...* **`]`**

DESCRIPTION

`zmore` is a filter which allows examination of compressed or plain text files, one screenful at a time on a soft-copy terminal. `zmore` works on files compressed with `compress`, `pack`, or `gzip`, and also on uncompressed files. If a file does not exist, `zmore` looks for a file of the same name with the addition of a `.gz`, `.z`, or `.Z` suffix.

`zmore` normally pauses after each screenful, printing **`--More--`** at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilities are enumerated later.

`zmore` looks in the file `/etc/termcap` to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines. To use a pager other than the default more, set environment variable `PAGER` to the name of the desired program, such as `less`.

Other sequences which may be typed when `more` pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to `1`):

`[`*i*`]<space>` Display *i* more lines (or another screenful if no argument is given).

`[`*i*`]CONTROL-d` Display 11 more lines. If *i* is given, then the scroll size is set to *i*.

`[`*i*`]d` Same as CONTROL-d.

`[`*i*`]z` Same as typing `<space>` except that *i*, if entered, becomes the new window size.

`[`*i*`]s` Skip *i* lines and print a screenful of lines.

[*i*]f	Skip *i* screenfuls and print a screenful of lines.
q or Q	Exit from `more`.
e or q	When the prompt **--More--(Next file:** *file***)** is printed, this command causes `zmore` to exit.
s	When the prompt **--More--(Next file:** *file***)** is printed, this command causes `zmore` to skip the next file and continue.
=	Display the current line number.
[*i*]/*expr*	Searches for the *i*-th occurrence of the regular expression *expr*. If the pattern is not found, `zmore` goes on to the next file, if any. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's `erase` and `kill` characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.
[*i*]*n*	Search for the *i*-th occurrence of the last regular expression entered.
!*command*	Invoke a shell with *command*. The characters `%` and `!` in *command* are replaced with the current file name and the previous shell command respectively. If there is no current file name, `%` is not expanded. The sequences `\%` and `\!` are replaced by `%` and `!`, respectively.
:q or **:Q**	Exit from `more` (same as `q` or `Q`).
`.` (dot)	Repeat the previous command.

The commands take effect immediately; it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being typed. In addition, the user may hit the erase character to redisplay the **--More--(xx%)** prompt.

When output is being sent to the terminal, the user can type the quit character. `more` will stop sending output, and will display the usual **--More--** prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when quit signal occurs.

The terminal is set to `noecho` mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the `/` and `!` commands.

If the standard output is not a teletype, then `zmore` acts just like `cat`, except that a header is printed before each file (if there is more than one).

FILES

`/etc/termcap`	Terminal data base

SEE ALSO

Utility Programs - `more, gzip, zdiff, zgrep, znew, zforce, gzexe`

NAME

`znew` – recompress .Z files to .gz files

SYNOPSIS

`znew [ -ftv9PK ] [` *name.Z ...***`]`**

DESCRIPTION

`znew` recompresses files from `.Z` (`compress`) format to `.gz` (`gzip`) format. If you want to recompress a file already in `gzip` format, rename the file to force a `.Z` extension then apply `znew`.

OPTIONS

`-f` Force recompression from `.Z` to `.gz` format even if a `.gz` file already exists.

`-t` Tests the new files before deleting originals.

`-v` Verbose. Display the name and percentage reduction for each file compressed.

`-9` Use the slowest compression method (optimal compression).

`-P` Use pipes for the conversion to reduce disk space usage.

`-K` Keep a `.Z` file when it is smaller than the `.gz` file

SEE ALSO

Utility Programs - `gzip, zmore, zdiff, zgrep, zforce, gzexe, compress`

BUGS

`znew` does not maintain the time stamp with the **`-P`** option.

Master Index

U

V

W

X-Z