NAME

telinit − user communication with init

SYNOPSIS

telinit [**0123456sSQabc**]

DESCRIPTION

Telinit, which is linked to /etc/**init**, is used to direct the actions of *init*. It takes a one character argument and signals *init* via the kill system call to perform the appropriate action. The following arguments serve as directives to *init*.

[0-6]	tells *init* to place the system in one of the run states **0-6**.
[a,b,c]	tells *init* to process only those /**etc/inittab** file entries having a the **a**, **b** or **c** run state set.
Q	tells *init* to reexamine the /**etc/inittab** file.
[s,S]	tells *init* to enter the single user environment. When this level change is effected, the virtual system teletype, /**dev/syscon**, is changed to the terminal from which the command was executed.

Telinit can only be run by one who is super user or a member of group sys.

FILES

init(1M), who(1), kill(2), inittab(5)

DIAGNOSTICS

Illegal Argument

NAME

test − condition evaluation command

SYNOPSIS

test expr
[expr]

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

−r *file*	true if *file* exists and is readable.
−w *file*	true if *file* exists and is writable.
−x *file*	true if *file* exists and is executable.
−f *file*	true if *file* exists and is a regular file.
−d *file*	true if *file* exists and is a directory.
−c *file*	true if *file* exists and is a character special file.
−b *file*	true if *file* exists and is a block special file.
−u *file*	true if *file* exists and its set-user-ID bit is set.
−g *file*	true if *file* exists and its set-group-ID bit is set.
−k *file*	true if *file* exists and its sticky bit is set.
−s *file*	true if *file* exists and has a size greater than zero.
−t [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
−z *s1*	true if the length of string *s1* is zero.
−n *s1*	true if the length of the string *s1* is non-zero.
s1 **=** *s2*	true if strings *s1* and *s2* are equal.
s1 **!=** *s2*	true if strings *s1* and *s2* are *not* equal.
s1	true if *s1* is *not* the null string.
n1 **−eq** *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, and **−le** may be used in place of **−eq**.

These primaries may be combined with the following operators:

!	unary negation operator.
−a	binary *and* operator.
−o	binary *or* operator (**−a** has higher precedence than **−o**).
(expr)	parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

SEE ALSO

find(1), sh(1).

WARNING

In the second form of the command (i.e., the one that uses [ ], rather than the word *test*), the square brackets must be delimited by blanks.

NAME

time − time a command

SYNOPSIS

time command [args ...]

DESCRIPTION

The *command* is executed with arguments *args*; after it completes, the following report is printed on the standard output:

real	*rtime*
user	*utime*
sys	*stime*
breads	*nbreads*
bwrites	*nbwrites*

rtime:	Total elapsed clock time during execution of *command*.
utime:	Time spent in *user* mode during execution of *command*.
stime:	Time spent in *system* mode during execution of *command*.
breads:	Number of block device reads (e.g., disk, magtape, etc.) caused by *command*.
bwrites:	Number of block device writes during execution of *command*.

The execution time can depend on what kind of memory the program happens to land in; the user time in MOS is often half what it is in core.

BUGS

Notice that:

```
time who >x
```

puts the timing information into *x*. If it desired to put the output from *who* into file *x* the following command line will serve:

```
time sh -c who >x
```

Elapsed time is accurate to the second, while the CPU times are measured to the 60th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

tk − paginator for the Tektronix 4014

SYNOPSIS

tk [**−t**] [−N] [**−pL**] [name]

DESCRIPTION

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

The command line options are:

−t Don't wait between pages; for directing output into a file.

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

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

SEE ALSO

nroff(1)

NAME

tkdump − prints a Tektronix file

USAGE

tkdump file [file ...]

DESCRIPTION

Tkdump prints a file containing Tektronix scope code, any file name ending with '**.tk**' written by *gex*(1G).

Example:

```
Wed Aug 24 14:12:00 1977    t.tk

Line Style=140
J252,442  V252,442  V536,442  V536,524
Line Style=150
J253,276  V253,276
J251,276  V251,276
J252,276  V252,276
J252,275  V536,275
J252,277  V536,277
J252,276  V536,276
J537,276  V537,214
J535,276  V535,214
J536,276  V536,214
```

SEE ALSO

gex(1G)

AUTHOR

D. J. Jackowski

NAME

tm − meditate

SYNOPSIS

tm [−number] [time]

DESCRIPTION

Tm causes UNIX to go into a state in which all current activities are suspended for *time* minutes (default is 20). At the beginning of this period, *tm* generates a set of *number* (default 3) transcendental numbers. Then it prints a two- to six-character nonsense syllable (*mantra*) on every logged-in terminal (a *different* syllable on each terminal). For the remainder of the time interval, it repeats these numbers to itself, in random order, binary digit by binary digit (memory permitting), while simultaneously contemplating its kernel. It is suggested users utilize the time thus provided to do some meditating (or praying) themselves. One possibility is to close one's eyes, attempt to shut out one's surroundings, and concentrate on the *mantra* supplied by *tm*. At the end of the time interval, UNIX returns to the suspended activities, refreshed and reinvigorated. Hopefully, so do the users.

FILES

Tm does not use any files, from external influences and distractions.

DIAGNOSTICS

If disturbed for any reason during the interval of meditation, *tm* locks the keyboard on every terminal, prints an unprintable expletive, and unlocks the keyboard. Subsequent UNIX operation may be marked by an unusual number of lost or scrambled files and droped lines.

BUGS

If *number* is greater that 32,767 (decimal), *tm* appears to generate *rational* numbers for the entire time interval, after which the behavior of the system may be completely *irrational* (i.e. transcendental).

WARNING

Attempts to use *flog*(1) on *tm* are invariably counterproductive.

NOTE:

This page was copied from PWB/UNIX Release 2.0 (IH) and brought to you for your amusement.

NAME

touch − change modification time of a file

SYNOPSIS

touch [−c] [−fprotofile] file ...

DESCRIPTION

Touch with no arguments, will cause the modification time of each *file* to be changed to current time.

The **−c** option prevents *touch* from creating the file if it did not previously exist.

The **−f** option tells *touch* to make the modification time of *file* the same as the modification time of *protofile*. There must be no blank between **−f** and *protofile*.

SEE ALSO

stat(2), time(2), utime(2),

DIAGNOSTICS

All diagnostics are hopefully self-explanatory.

BUGS

The access time is unchanged intentionally.
Future enhancements to accept date format *mmddhhmm*[*yy*]

NAME

tp − manipulate tape archive

SYNOPSIS

tp [key] [name ...]

DESCRIPTION

Tp saves and restores files on DECtape or magtape. Its actions are controlled by the *key* argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named files are written on the tape. If files with the same names already exist, they are replaced. 'Same' is determined by string comparison, so **./abc** can never be the same as **/usr/sbo/abc** even if **/usr/sbo** is the current directory. If no file argument is given, . is the default.

u updates the tape. **u** is like **r**, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. **u** is the default command if none is given.

d deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magtapes.

x extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.

t lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

m Specifies magtape as opposed to DECtape.

b Normally *tp* writes a boot routine in tape block 0 and begins writing the directory and specified files in block 1. The **b** modifier causes **tp** to write the boot routine in block 1 and begin writing the directory and specified files in block 2. This modifier is used in place of **m** for writing a DEC ROM bootable magtape.

0,...,7 This modifier selects the drive on which the tape is mounted. For DECtape, **x** is default; for magtape **0** is the default.

v Normally *tp* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

c means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with **r** and **u.** This option is assumed with magtape since it is impossible to selectively overwrite magtape.

i Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

f Use the first named file, rather than a tape, as the archive. This option is known to work only with **x**.

w causes *tp* to pause before treating each file, type the indicative letter and the file name (as with **v**) and await the user's response. Response **y** means 'yes', so the file is treated. Null response means 'no', and the file does not take part in whatever is being done. Response **x** means 'exit'; the *tp* command terminates immediately. In the **x** function, files previously asked about have been extracted already. With **r, u,** and **d** no change has been made to the tape.

FILES

/dev/tap?
/dev/mt?

SEE ALSO

ar(1), tar(1)

DIAGNOSTICS

Several; the non-obvious one is 'Phase error', which means the file changed after it was selected for dumping but before it was dumped.

BUGS

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by *tp* difficult to carry to other machines; *tar*(1) avoids the problem.

NAME

tr − translate characters

SYNOPSIS

tr [**−cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options **−cds** may be used: **−c** complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal; **−d** deletes all input characters in *string1;* **−s** squeezes all strings of repeated output characters that are in *string2* to single characters.

In either string the notation *a*−*b* means a range of characters from *a* to *b* in increasing ASCII order. The character \ followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A \ followed by any other character stands for that character.

EXAMPLE

The following example creates a list of all the words in **file1** one per line in **file2**, where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect \ from the Shell. 012 is the ASCII code for new-line.

```
tr −cs A−Za−z ′\012′ <file1 >file2
```

SEE ALSO

ed(1), ascii(7)

BUGS

Won't handle ASCII NUL in *string1* or *string2;* always deletes NUL from input.

NAME

true, false — provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true
do
        command
done
```

SEE ALSO

sh(1)

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tset - set terminal modes

SYNOPSIS

tset [options] [**-m** [ident][test baudrate]:type ...] [type]

DESCRIPTION

Tset causes terminal dependent processing such as setting erase and kill characters, setting or resetting delays, and the like. It first determines the *type* of terminal involved, names for which are specified by the /*etc*/*termcap* data base, and then does necessary initializations and mode settings. In the case where no argument types are specified, *tset* simply reads the terminal type out of the environment variable TERM and re-initializes the terminal. The rest of this manual concerns itself with type initialization, done typically once at login, and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script .*profile* (for sh(1) users) or .*login* (for *csh*(1) users) it is desirable to give information about the types of terminal usually used on terminals which are not hardwired. These ports are initially identified as being *dialup* or *plugboard* or *arpanet* etc. To specify what terminal type is usually used on these ports **-m** is followed by the appropriate port type identifier, an optional baud-rate specification, and the terminal type to be used if the mapping conditions are satisfied. If more than one mapping is specified, the first applicable mapping prevails. A missing type identifier matches all identifiers.

Baud rates are specified as with *stty*(1), and are compared with the speed of the diagnostic output (which is almost always the control terminal). The baud rate test may be any combination of: **>**, **=**, **<**, **@**, and **!**; **@** is a synonym for **=** and **!** inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to **-m** within ``'' characters; users of *csh*(1) must also put a ``\'' before any ``!'' used here.

Thus

```
     tset    -m    'dialup>300:adm3a'     -m     dialup:dw2     -m
     'plugboard:?adm3a'
```

causes the terminal type to be set to an *adm3a* if the port in use is a dialup at a speed greater than 300 baud; to a *dw2* if the port is (otherwise) a dialup (i.e. at 300 baud or less). If the *type* above begins with a question mark, the user is asked if s/he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. Thus, in this case, the user will be queried on a plugboard port as to whether they are using an *adm3a*. For other ports

the port type will be taken from the /etc/ttytype file or a final, default *type* option may be given on the command line not preceded by a **-m**.

It is often desirable to return the terminal type, as specified by the **-m** options, and information about the terminal to a shell's environment. This can be done using the **-s** option; using the Bourne shell, *sh*(1):

```
eval `tset -s options...`
```

or using the C shell, *csh*(1):

```
set noglob; eval `tset -s options...`
```

These commands cause *tset* to generate as output a sequence of shell commands which place the variables TERM and TERMCAP in the environment; see *environ*(5).

Once the terminal type is known, *tset* engages in terminal mode setting. This normally involves sending an initialization sequence to the terminal and setting the single character erase (and optionally the line-kill (full line erase)) characters.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character (`#' on standard systems), the erase character is changed to a Control-H (backspace).

The options are:

-e set the erase character to be the named character *c* on all terminals, the default being the backspace character on the terminal, usually ^H.

-k is similar to **-e** but for the line kill character rather than the erase character; *c* defaults to ^X (for purely historical reasons); ^U is the preferred setting. No kill processing is done if **-k** is not specified.

-I supresses outputting terminal initialization strings.

-Q supresses printing the ``Erase set to'' and ``Kill set to'' messages.

-S Outputs the strings to be assigned to TERM and TERMCAP in the environment rather than commands for a shell.

FILES

/etc/ttytype	terminal id to type map database
/etc/termcap	terminal capability database

SEE ALSO

csh(1), setenv(1), sh(1), stty(1), environ(5), ttytype(5), termcap(5)

AUTHOR

Eric Allman

BUGS

Should be merged with _stty_(1).

NOTES

For compatibility with earlier versions of _tset_ a number of flags are accepted whose use is discouraged:

-d type	equivalent to **-m** dialup:type
-p type	equivalent to **-m** plugboard:type
-a type	equivalent to **-m** arpanet:type
-E c	Sets the erase character to _c_ only if the terminal can backspace.
-	prints the terminal type on the standard output
-r	prints the terminal type on the diagnostic output.

NAME

tsort − topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

ttt — tic-tac-toe

SYNOPSIS

/usr/games/ttt

DESCRIPTION

Ttt is the X and O game popular in the first grade. This is a learning program that never makes the same mistake twice.

Although it learns, it learns slowly. It must lose nearly 80 games to completely know the game.

FILES

/usr/games/ttt.k	learning file

NAME

tty − get the terminal's name

SYNOPSIS

tty [**−s**]

DESCRIPTION

Tty prints the pathname of the user's terminal. The **−s** option inhibits printing, allowing one to just test the exit code.

EXIT CODES

0	if standard input is a terminal
1	otherwise.

DIAGNOSTICS

"not a tty" if the standard input file is not a terminal (and **−s** is not specified).

NAME

turbo — encabulator

DESCRIPTION

For a number of years now work has been proceeding in order to bring perfection to the crudely conceived idea of a machine that would not only supply inverse reactive current for use in unilateral phase detractors, but would also be capable of automatically synchronizing cardinal grammeters. Such a machine is the "Turbo-Encabulator". Basically, the only new priniple involved is that instead of power being generated by the relaxive motion of conductors and fluxes, it is produced by the modial interaction of megneto-reluctance and capacitive directence.

The original machine had a base-plate of prefabulated amulite, surmounted by a malleable logarithmic casing in such a way that the two spurving bearings were in a direct line with the pentametric fan. The latter consisted simply of six hydrocoptic marzelvanes, so fitted to the ambifacient lunar waneshaft that sides fumbling was effectively prevented. The main winding was of the normal lotus-o-delta type placed in panendermic semiboloid slots in the stator, every seventh conductor being connected by a non-reversible tremie pipe to the differential girdlespring on the "up" end of the grammeters.

Forty-one manestically spaced grouting brushes were arranged to feed into the rotor slip-stream a mixture of high S-valve phenyhydrobenzamine and five per cent reminative tetryliodohexamine. Both of these liquids have specific pericosities given by P-2.5Cn where n is the diathetical evolute of retrograde temperature phase disposition and C is the Cholmondeley's anular grillage coefficient. Initially, n was measured with the aid of a metapolar refractive pilfrometer (for a description of this ingenious instrument, see L. E. Rumpelverstein in "Zeitschrift fur Elektrotechnistiatischs-Donnerblitze", vol. vii.), but up to the present date nothing has been found to equal the transcendental hopper dadoscope (see "Prological Sciences," June, 1914).

Electrical engineers will appreciate the difficulty of nubing together a regurgitative purwell and a supramitive wennelsprocket. Indeed, this proved to be a stumbling block to further development until, in 1942, it was found that the use of anhydrous nagling pins enabled a kyptonastic bolling shim to be tankered.

The early attempts to construct a sufficiently robust spiral decommutator failed largely because of a lack of appreciation of the large quasi-piestic stress in the gremlin studs; the latter were specially designed to hold the roffit bars to the spamshaft. When, however, it was discovered that wending could be prevented by a simple addition to the living sockets almost perfect running was secured.

The operating point is maintained as near as possible to the h.f. rem peak by constantly fromaging the bitumogenous spandrels. This is a distinct advance on the standard nivelsheave in that no dramcock oil is required after the phase detractors have remissed.

Undoubtedly, the turbo-encabulator has now reached a very high level of technical development. It has been successfully used for operating nofer trunnions. In addition whenever a barescent skor motion is required, it may be employed in conjunction with the drawn reciprocating dingle arm to reduce sinusoidal depleneration.

DIAGNOSTICS

All diagnostics are printed on file descriptor 2.

BUGS

The living sprockets can sometime react unfavorably with the hydrocoptic marzelvanes to produce a high level of radiation. This should not be considered a problem though.

NAME

typo − find possible typos

SYNOPSIS

typo [**−n**] [file] ...

DESCRIPTION

Typo hunts through a document for unusual words, typographic errors, and *hapax legomena* and prints them on the standard output.

The words used in the document are printed out in decreasing order of peculiarity along with an index of peculiarity. An index of 10 or more is considered peculiar. Printing of certain very common English words is suppressed.

The statistics for judging words are taken from the document itself, with some help from known statistics of English. The **−n** option suppresses the help from English and should be used if the document is written in, for example, Urdu.

Troff(1) control lines are ignored. Quote marks, vertical bars, hyphens, and ampersands within words are equivalent to spaces. Words hyphenated across lines are put back together.

FILES

/tmp/ttmp??
/usr/lib/salt
/usr/lib/sq2006

SEE ALSO

spell(1).

NAME

ucore − turn on or off the unique core dumping feature.

SYNOPSIS

ucore on|off shell command

DESCRIPTION

Ucore is used to turn **on** and **off** the unique core dumping feature of the operating system. When the feature is **off** (the default), core files are dumped in **core**. When the feature is enabled, cores are dumped in **core.***nnnnn*, where *nnnnn* is the process id of the process that terminated with the core dump. This feature is useful when more than one process drops a core, or when it is necessary to terminate a multiprocess action and it is desired to see what each process was doing. To get a *sh* which forks processes with the unique core feature on, do the following:

```
exec ucore on sh
```

This will replace the current shell with one in which the feature is enabled.

SEE ALSO

ucore(2)

NAME
umount − dismount file system

SYNOPSIS
umount special ...

DESCRIPTION
Umount announces to the system that the removable file system(s) previously mounted on special file *special* is/are to be removed.

SEE ALSO
mount(1), umount(2), mtab(5)

FILES
/etc/mtab mounted device table

DIAGNOSTICS
Frequently, in the case of **/usr**, the file system is found to be busy because one of the Shell accounting files (**/usr/adm/sh_acct** or **/usr/adm/su_acct**) is open for some Shell or system accounting is active (**/usr/adm/acct**).

NAME
uname − print name of current UNIX

SYNOPSIS
uname [**−snrv**]

DESCRIPTION
Uname prints the current name of UNIX on the standard output file. It is mainly useful to determine what system one is using. The options cause selected information returned by *uname*(2) to be printed:

−s	print the system name (default).
−n	print the nodename (the nodename may be a name that the system is known by to a communications network i.e. *uucp*(1C).)
−r	print the operating system release.
−v	print the operating system version.

SEE ALSO
uname(2)

NAME

unhex − translate hexed file to binary

SYNOPSIS

unhex file1 file2

DESCRIPTION

Unhex will read file1 outputting into file2 one byte for every two characters read. The input is assumed to be in *hex* format (i.e., that produced by the *hex* program).

SEE ALSO

cu(1C), hex(1)

DIAGNOSTICS

Argument count
Unable to open input file
Unable to create output file
Unexpected line feed
Garbage in file

NAME

uniq − report repeated lines in a file

SYNOPSIS

uniq [**−udc** [**+**n] [**−**n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **−u** flag is used, just the lines that are not repeated in the original file are output. The **−d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **−u** and **−d** mode outputs.

The **−c** option supersedes **−u** and **−d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

−*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), sort(1)

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.54000e+00
        / 3.93701e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 pounds force/in2
You want: atm
        * 1.02069e+00
        / 9.79730e-01
```

Units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

'Pound' is a unit of mass. Compound names are run together, e.g. 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. For a complete list of units, 'cat /usr/lib/unittab'.

FILES

/usr/lib/unittab

NAME

unpack − expand compressed files

SYNOPSIS

unpack name ...

DESCRIPTION

Unpack expands files created by *pack*(1). For each file *name* specified in the command, a search is made for a file called *name***.z** (or just *name*, if *name* ends in **.z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur if:

> the file name (exclusive of the **.z**) has more than 12 characters;
> the file cannot be opened;
> the file does not appear to be the output of *pack*(1);
> a file with the "unpacked" name already exists;
> if the unpacked file cannot be created.

SEE ALSO

pack(1), pcat(1).

NAME

update − periodically update the super block

SYNOPSIS

update

DESCRIPTION

Update is a program that executes the *sync* primitive every 60 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the **/etc/lines** file. See *sync (2)* for details.

SEE ALSO

init(1M), sync(2)

BUGS

With *update* running, if the CPU is halted just as the *sync* is executed, a file system can be damaged. This is partially due to DEC hardware that writes zeros when NPR requests fail. A fix would be to have *sync* temporarily increment the system time by at least 60 seconds to trigger the execution of *update.* This would give 60 seconds grace to halt the CPU.

NAME

updfs − update file system

SYNOPSIS

/etc/updfs [−Nmxctviogfuebpq R] [tapefile] [name] [file] [dir]

DESCRIPTION

Updfs reads magtapes (or files) created by the *cmpfs* command and updates the file system named by the *dir* argument. Its actions are governed by the specified flags (in the absence of all flags, *updfs* simply lists the contents of the tape):

N *N* is an optional one or two decimal digits which designate which mag tape drive the program should use. Drive 0 is default if no drive is specified.

m Use the file *tapefile* as the input instead of a mag tape drive.

x Extract files from the tape and write them into the file system as given by their name on the tape. Names on the tape are relative, so that all files extracted from the tape will end up being heirarchically lower than the argument *dir* .

c Compare files on the tape with the corresponding files in the file system under the argument *dir*. Only the files which are different are noted. If the *x* flag has also been specified, *updfs* will not extract a file unless it mismatches. Note that the only difference between *x* and *xc* is that the modification dates on all files will be changed in the former case, while the mod dates on only mismatching files will be changed in the latter. Files may mismatch in terms of mode, owner id, group id, file contents, or, in the case of special files, their major/minor device assignment.

t Produce list of files for which *updfs* did something. This list will contain a subset of all the names on the tape only if the *c* , *i* or *o* flags are specified. Each line will consist of the entry type [dcal], the relative path name, and the link-to path name if the entry is a link entry.

v Produce verbose output. In addition to the information printed when the *t* flag is given, the verbose option expands the output to include the mode, link count, owner id, group id, and size of the file on the tape. In addition, if the *c* flag was specified, another line of output is generated to show the mode, ownership, etc, of the corresponding file on the file system, if any. This helps to identify exactly why *updfs* thought the two files mismatched. If the two lines of output are identical, then it may be assumed that the mismatch occurred in the contents of the files.

i Ignore all files on the tape which have the same relative pathname as one of the pathnames in an ignore file, or which are heirarchically lower. The name of the ignore file is taken to be the next argument in the argument list (i.e. *name* above).

o Look at only files which match a name in an only file or are heirarchically lower. The name of the only file is taken to be the next argument in the argument list. An ignore/only file should be a list of relative pathnames (both file names and directory names are allowed) separated by newlines. The "relative" requirement is important; for example, it should be clear that no pathname may start with a "/". Although it is logically possible to have a situation where it would be convenient to have both an ignore and an only file, *updfs* allows only one or the other to be used.

g *updfs* is to look on the tape for the single file name given by the following argument (*name*). If the name turns out to be a directory name, only the directory name is examined; heirarchically lower names are not examined. The flags *x*, *c*, *t* and *v* apply, but the flags *i* and *o* are not allowed. The purpose of this option is to bypass the "only" option if only one name is to be examined on the tape. Note that if either the *x* or *c* flag is given, the *dir* argument must also be given.

f Useful only if the flags *x* and *g* have been specified. This flag causes the single file (*name*) to be extracted from the tape and given the name *file* .

u Unlink any file about to be created. Useful only when the *x* flag has been specifed, this option allows an update to be brought in as the specified file name, but destroys any previous links to the file. If the unlink fails, either because the user does not have the proper permission or because the file does not previously exist, a create failure message will be printed.

e May be used when the tape is an incremental backup (created with the *e* option in the cmpfs command). Any files or directories in the file system which were not in existence when the incremental was made will be deleted (if *x* is specified) and listed (if *t* or *v* is specified). Using this option with updfs after reloading an epoch tape will restore the file system to its state when the cmpfs was performed.

b Input is blocked 5120 blocks per record instead of 512.

p ignore file *mode* when doing comparsion.

q ignore file *user* ownership when doing comparsion.

R ignore file *group* ownership when doing comparsion.

FILES

/dev/mtN

SEE ALSO

cmpfs(1M)

NAME

uuclean − uucp spool directory clean-up

SYNOPSIS

uuclean [options] ...

DESCRIPTION

Uuclean will scan the spool directory for files with the specified prefix and delete all those which are older than the specified number of hours.

The following options are available.

−**d***directory*
Clean *directory* instead of the spool directory.

−**p***pre* Scan for files with *pre* as the file prefix. Up to 10 −**p** arguments may be specified. A −**p** without any *pre* following will cause all files older than the specified time to be deleted.

−**n***time* Files whose age is more than *time* hours will be deleted if the prefix test is satisfied. (default time is 72 hours)

−**m** Send mail to the owner of the file when it is deleted.

This program will typically be started by *cron*(1M).

FILES

/usr/lib/uucp	directory with commands used by *uuclean* internally
/usr/spool/uucp	spool directory

SEE ALSO

uucp(1C), uux(1C).

NAME

uucp, uulog, uuname — unix to unix copy

SYNOPSIS

uucp [option] ... source-file ... destination-file

uulog [option] ...

uuname [−l]

DESCRIPTION

Uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names which *uucp* knows about. Shell metacharacters ?*[] appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

(1) a full path name;

(2) a path name preceded by ˜*user* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by ˜/*user* where *user* is a login name on the specified system and is replaced by that user's directory under PUBDIR;

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

The following options are interpreted by *uucp*:

−d Make all necessary directories for the file copy (default).

−f Do not make intermediate directories for the file copy.

−c Use the source file when copying out rather than copying the file to the spool directory (default).

−C Copy the source file to the spool directory.

−m Send mail to the requester when the copy is complete.

−n*user*
Notify *user* on the remote system that a file was sent.

−e*sys* Send the *uucp* command to system *sys* to be executed there. (Note — this will only be successful if the remote machine allows the *uucp* command to be executed by **/usr/lib/uucp/uuxqt**.)

Uulog maintains a summary log of *uucp* and *uux*(1C) transactions in the file **/usr/spool/uucp/LOGFILE** by gathering information from partial log files named **/usr/spool/uucp/LOG.*.?**. (These files will only be created if the **LOGFILE** is being used by another process.) It removes the partial log files.

The options cause *uulog* to print logging information:

−s*sys* Print information about work involving system *sys*.

−u*user*
Print information about work done for the specified *user*.

Uuname lists the uucp names of known systems. The **−l** option returns the local system name.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory for receiving and sending (PUBDIR)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uux(1C).
Uucp Implementation Description by D. A. Nowitz.

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to **~nuucp** or just **~**).

BUGS

All files received by *uucp* will be owned by *uucp*.
The **−m** option will only work sending files or receiving a single file. (Receiving multiple files specified by special shell characters **?*[]** will not activate the **−m** option.)

NAME

uulog − uucp user log inquiry

SYNOPSIS

uulog [option] ...

DESCRIPTION

Uulog will search through the uucp log file and output the requested lines.

The requested lines are specified using one or more of the following options:

−s*sys* Output lines which have *sys* as a prefix to the system name;

−u*user* Output lines which have *user* as a prefix to the user who requested the work.

FILES

/usr/lib/uucp - directory with commands used by *uulog* internally
/usr/lib/uucp/spool - spool directory
/usr/lib/uucp/spool/LOGFILE

SEE ALSO

uucp(1C), uux(1C), uuclean(1C)

NAME

uunames – list names of UNIX systems known to uucp

SYNOPSIS

uunames [**–l**] [**–v**]

DESCRIPTION

Uunames returns the list of the names of all UNIX systems known to the local *uucp*(1C). The **–l** option causes *uunames* to return the name of the local system. The **–v** option causes *uunames* to return the description of the uucp systems.

FILES

/usr/lib/uucp/L.sys
/usr/lib/uucp/ADMIN
/usr/lib/uucp/uuname

SEE ALSO

uucp(1C)

NAME

uurec - receive processed news articles via mail

SYNOPSIS

uurec

DESCRIPTION

uurec reads news articles on the standard input sent by *send-news*(1), decodes them, and gives them to *inews*(1) for insertion.

SEE ALSO

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

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [option] ...

DESCRIPTION

Uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. The following options are recognized:

–**m***mch*	Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local *uucp* are provided.
–**k***jobn*	Kill the *uucp* request whose job number is *jobn*. The killed *uucp* request must belong to the person issuing the *uustat* command unless he is the super-user.
–**c***hour*	Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the super-user.
–**u***user*	Report the status of all *uucp* requests issued by *user*.
–**s***sys*	Report the status of all *uucp* requests which communicate with remote system *sys*.
–**o***hour*	Report the status of all *uucp* requests which are older than *hour* hours.
–**y***hour*	Report the status of all *uucp* requests which are younger than *hour* hours.
–**j***all*	Report the status of all the *uucp* requests.
–**v**	Report the *uucp* status verbosely. If this option is not specified, a status code is printed with each *uucp* request.

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user. Note that only one of the options **–j**, **–m**, **–k**, **–c**, or the rest of other options may be specified.

For example, the command

uustat –ujdd –scbosg –y72 –v

will print the verbose status of all *uucp* requests that were issued by user *jdd* to communicate with system *cbosg* within the last 72 hours. The meanings of the job request status are:

job-number user remote-system command-time status-time status

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
00001	the copy failed, but the reason cannot be determined
00002	permission to access local file is denied
00004	permission to access remote file is denied
00010	bad *uucp* command is generated
00020	remote system cannot create temporary file
00040	cannot copy to remote directory
00100	cannot copy to local directory
00200	local system cannot create temporary file
00400	cannot execute *uucp*
01000	copy succeeded
02000	copy finished, job deleted
04000	job is queued

The meanings of the machine accessibility status are:

system-name time status

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucp/L_stat	system status file
/usr/spool/uucp/R_stat	request status file

SEE ALSO

uucp(1C).
Uustat − A UUCP Status Inquiry Program, by H. Che.

NAME

uusub — monitor uucp network

SYNOPSIS

uusub [options]

DESCRIPTION

Uusub defines a *uucp* subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

−a*sys*	Add *sys* to the subnetwork.
−d*sys*	Delete *sys* from the subnetwork.
−l	Report the statistics on connections.
−r	Report the statistics on traffic amount.
−f	Flush the connection statistics.
−u*hr*	Gather the traffic statistics over the past *hr* hours.
−c*sys*	Exercise the connection to the system *sys*. If *sys* is specified as **all**, then exercise the connection to all the systems in the subnetwork.

The meanings of the connections report are:

```
sys #call #ok time #dev #login #nack #other
```

where *sys* is the remote system name, *#call* is the number of times the local system tries to call *sys* since the last flush was done, *#ok* is the number of successful connections, *time* is the the latest successful connect time, *#dev* is the number of unsuccessful connections because of no available device (e.g. ACU), *#login* is the number of unsuccessful connections because of login failure, *#nack* is the number of unsuccessful connections because of no response (e.g. line busy, system down), and *#other* is the number of unsuccessful connections because of other reasons.

The meanings of the traffic statistics are:

```
sfile sbyte rfile rbyte
```

where *sfile* is the number of files sent and *sbyte* is the number of bytes sent over the period of time indicated in the latest *uusub* command with the **−u***hr* option. Similarly, *rfile* and *rbyte* are the numbers of files and bytes received.

The command:

```
uusub −c all −u 24
```

is typically started by *cron*(1M) once a day.

FILES

/usr/spool/uucp/SYSLOG	system log file
/usr/lib/uucp/L_sub	connection statistics
/usr/lib/uucp/R_sub	traffic statistics

SEE ALSO

uucp(1C), uustat(1C).

NAME

uux − unix to unix command execution

SYNOPSIS

uux [−] command-string

DESCRIPTION

Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail*(1)) via *uux*.

The *command-string* is made up of one or more arguments that look like a Shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

(1) a full path name;

(2) a path name preceded by ˜*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

The − option will cause the standard input to the *uux* command to be the standard input to the *command-string*. For example, the command

```
uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff"
```

will get the **f1** files from the "usg" and "pwba" machines, execute a *diff* command and put the results in **f1.diff** in the local directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

Uux will attempt to get all files to the execution system. For files which are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!uucp b!/usr/file \(c!/usr/file\)
```

will send a *uucp* command to system "a" to get **/usr/file** from system "b" and send it to system "c".

Uux will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine.

FILES

/usr/lib/uucp/spool	spool directory
/usr/lib/uucp/*	other data and programs

SEE ALSO

uuclean(1M), uucp(1C).
Uucp Implementation Description by D. A. Nowitz

BUGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

NAME

val − validate SCCS file

SYNOPSIS

val −
val [−s] [−rSID] [−mname] [−ytype] file ...

DESCRIPTION

Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a −, and named files.

Val has a special argument, −, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

−s	The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.
−r*SID*	The argument value **SID** (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the **SID** is ambiguous (e. g., **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (e. g., **r**1.0 or **r**1.1.0 are invalid because neither case can exist as a valid delta number). If the **SID** is valid and not ambiguous, a check is made to determine if it actually exists.
−m*name*	The argument value *name* is compared with the SCCS %M% keyword in *file*.
−y*type*	The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = can't open file or file not SCCS;
bit 4 = **SID** is invalid or ambiguous;
bit 5 = **SID** does not exist;
bit 6 = %Y%, −y mismatch;
bit 7 = %M%, −m mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned − a logical OR of the codes generated for each command line and file processed.

SEE ALSO

admin(1S), delta(1S), get(1S), prs(1S)

DIAGNOSTICS

Use *help*(1S) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a *core* dump.

NAME

vpmc − compiler for the virtual protocol machine

SYNOPSIS

vpmc [**−m**] [**−r**] [**−c**] [**−x**] [**−s** sfile] [**−l** lfile] [**−i** ifile] [**−o** ofile] file

DESCRIPTION

Vpmc is the compiler for a language that is used to describe communications link protocols. The output of *vpmc* is a load module for the virtual protocol machine (VPM), which is a software construct for implementing communications link protocols (e.g., BISYNC) on the DEC KMC11-B microprocessor. VPM is implemented by an interpreter in the KMC which cooperates with a driver in the UNIX host computer to transfer data over a communications link in accordance with a specified link protocol. UNIX user processes transfer data to or from a remote terminal or computer system through VPM using normal UNIX *open*, *read*, *write*, and *close* operations. The VPM program in the KMC provides error control and flow control using the conventions specified in the protocol.

The language accepted by *vpmc* is essentially a subset of C; the implementation of *vpmc* uses the RATFOR preprocessor (*ratfor*(1)) as a front end; this leads to a few minor differences, mostly syntactic.

There are two versions of the interpreter. The appropriate version for a particular application is selected by means of the **−i** option. The BISYNC version (**−i bisync**) supports half-duplex, character-oriented protocols such as the various forms of BISYNC. The HDLC version (**−i hdlc**, the default) supports full-duplex, bit-oriented protocols such as HDLC. The communications primitives used with the BISYNC version are character-oriented and blocking; the primitives used with the HDLC version are frame-oriented and non-blocking.

Options

The meanings of the command-line options are:

−m	Use *m4*(1) instead of *cpp* as the macro preprocessor.
−r	Produce RATFOR output on the standard output and suppress the remaining compiler phases.
−c	Compile only (suppress the assembly and linking phases).
−x	Retain the intermediate files used for communication between passes.
−s *sfile*	Save the generated VPM assembly language on file *sfile*.
−l *lfile*	Produce a VPM assembly-language listing on file *lfile*.
−i *ifile*	Use the interpreter version specified by *ifile* (default **hdlc**).
−o *ofile*	Write the executable object file on file *ofile* (default **a.out**).

These options may be given in any order.

Programs

Input to *vpmc* consists of a (possibly null) sequence of array declarations, followed by one or more function definitions. The first defined function is invoked (on command from the UNIX VPM driver) to begin program execution.

Functions

A function definition has the following form:

```
function name()
statement_list
end
```

Function arguments (formal parameters) are not allowed. The effect of a function call with arguments can be obtained by invoking the function via a macro that first assigns the value of each argument to a global variable reserved for that purpose. See *EXAMPLES* below.

A *statement_list* is a (possibly null) sequence of labeled statements. A *labeled_statement* is a statement preceded by a (possibly null) sequence of labels. A *label* is either a name followed by a colon (:) or a decimal integer optionally followed by a colon.

The statements that make up a statement list must be separated by semicolons (;). (A semicolon at the end of a line can usually be omitted; refer to the description of RATFOR for details.) Null statements are allowed.

Statement Syntax

The following types of statements are allowed:

> *expression*
> *lvalue* = *expression*
> *lvalue* + = *expression*
> *lvalue* − = *expression*
> *lvalue* |= *expression*
> *lvalue* &= *expression*
> *lvalue* ^= *expression*
> *lvalue* <<= *expression*
> *lvalue* >>= *expression*
> if(*expression*)*statement*
> if(*expression*)*statement* else *statement*
> while(*expression*)*statement*
> for(*statement*; *expression*; *statement*)*statement*
> repeat *statement*
> repeat *statement* until *expression*
> break
> next
> switch(*expression*){*case_list*}
> return(*expression*)
> return
> goto *name*
> goto *decimal_constant*
> {*statement_list*}

repeat is equivalent to the **do** keyword in C; **next** is equivalent to **continue**.

A *case_list* is a sequence of statement lists, each of which is preceded by a label of the form:

> case *constant*:

The label for the last *statement_list* in a *case_list* may be of the form:

> default:

Unlike C, RATFOR supplies an automatic **break** preceding each new case label.

Expression Syntax

A *primary_expression* (abbreviated *primary*) is an lvalue or a constant. An *lvalue* is one of the following:

> *name*
> *name*[*constant*]

A *unary_expression* (abbreviated *unary*) is one of the following:

> *primary*
> *name*( )

> *system_call*
> ++*lvalue*
> −−*lvalue*
> (*expression*)
> !*unary*
> ˜*unary*

The following types of expressions are allowed:

> *unary*
> *unary*+*primary*
> *unary*−*primary*
> *unary*|*primary*
> *unary*&*primary*
> *unary*&˜*primary*
> *unary*^*primary*
> *unary*<<*primary*
> *unary*>>*primary*
> *unary*==*primary*
> *unary*!=*primary*
> *unary*>*primary*
> *unary*<*primary*
> *unary*>=*primary*
> *unary*<=*primary*

Note that the right operand of a binary operator can only be a constant, a name, or a name with a constant subscript.

System Calls

A VPM program interacts with a communications device and a driver in the host computer by means of system calls (primitives).

The following primitives are available only in the BISYNC version of the interpreter:

atoe(*primary*)
: Translate ASCII to EBCDIC. The returned value is the EBCDIC character that corresponds to the ASCII character represented by the value of the primary expression. The translation tables reflect the prejudices of a particular installation.

crc16(*primary*)
: The value of the primary expression is combined with the cyclic redundancy check-sum at the location passed by a previous **crcloc** system call. The CRC-16 polynomial ($x^{16}+x^{15}+x^2+1$) is used for the check-sum calculation.

crcloc(*name*)
: The two-byte array starting at the location specified by *name* is cleared. The address of the array is recorded as the location to be updated by subsequent **crc16** system calls.

etoa(*primary*)
: Translate EBCDIC to ASCII. The returned value is the ASCII character that corresponds to the EBCDIC character represented by the value of the primary expression. The translation tables reflect the prejudices of a particular installation.

get(*lvalue*)
: Get a byte from the current *transmit* buffer. The next available byte, if any, is copied into the location specified by *lvalue*. The returned value is zero if a byte was obtained, otherwise it is non-zero.

getrbuf(*name*)

Get (open) a *receive* buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a receive buffer has previously been opened via a **getrbuf** call but has not yet been closed via a call to **rtnrbuf**, that buffer is reinitialized and remains the current buffer.

getxbuf(*name*)
Get (open) a *transmit* buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a transmit buffer has previously been opened via a **getxbuf** call but has not yet been closed via a call to **rtnxbuf**, that buffer is reinitialized and remains the current buffer.

put(*primary*)
Put a byte into the current *receive* buffer. The value of the primary expression is inserted into the next available position, if any, in the current receive buffer. The returned value is zero if a byte was transferred, otherwise it is non-zero.

rcv(*lvalue*)
Receive a character. The process delays until a character is available in the input silo. The character is then moved to the location specified by *lvalue* and the process is reactivated.

rsom(*constant*)
Skip to the beginning of a new *receive* frame. The receiver hardware is cleared and the value of *constant* is stored as the receive sync character. This call is used to synchronize the local receiver and remote transmitter when the process is ready to accept a new receive frame.

rtnrbuf(*name*)
Return a *receive* buffer. The original values of the buffer parameters for the current receive buffer are replaced with values from the array specified by *name*. The current receive buffer is then released to the driver.

rtnxbuf(*name*)
Return a *transmit* buffer. The original values of the buffer parameters for the current transmit buffer are replaced with values from the array specified by *name*. The current transmit buffer is then released to the driver.

xeom(*constant*)
Transmit end-of-message. The value of the constant is transmitted, then the transmitter is shut down.

xmt(*primary*)
Transmit a character. The value of the primary expression is transmitted over the communications line. If the output silo is full, the process waits until there is room in the silo.

xsom(*constant*)
Transmit start-of-message. The transmitter is cleared, then the value of *constant* is transmitted six times. This call is used to synchronize the local transmitter and the remote receiver at the beginning of a frame.

The following primitives are available only with the HDLC version of the interpreter:

abtxfrm( )
The current transmission, if any, is aborted, if possible, by sending a frame-abort

sequence (seven one bits, followed immediately by a terminating flag). This operation is not feasible with some hardware interfaces, in which case this primitive is a no-operation.

getxfrm(*primary*)
: Get a transmit buffer. If the transmit-buffer queue is *not* empty, the buffer at the head of the queue is removed from the queue and attached to the sequence number specified by the value of the primary expression If the sequence number is greater than seven or the sequence number already has a buffer attached, the process is terminated in error. The returned value is zero if a buffer was obtained, otherwise non-zero.

norbuf()
: Test for the availability of an empty receive buffer. The returned value is **true** (non-zero) if the queue of empty receive buffers is currently empty; otherwise the returned value is **false** (zero).

rcvfrm(*name*)
: Get a completed receive frame. If the queue of completed receive frames is non-empty, the frame at the head of the queue is removed and becomes the current receive frame. If a frame is obtained, the first five bytes of the frame are copied into the array specified by *name*. The returned value is **true** (non-zero) if a frame was obtained; otherwise, it is **false** (zero). The rightmost four bits of the returned value indicate the frame length as follows: if the value of the rightmost four bits is equal to fifteen, the frame length is greater than or equal to 15; otherwise the frame length is equal to the value of the rightmost four bits. The frame length includes the two CRC bytes at the end of the frame and any control information at the beginning of the frame. Bytes following the first two bytes of the frame, but not including the two CRC bytes, are copied into a receive buffer, if one is available at the time the frame is received. Bit 020 of the returned value is zero if a receive buffer was available, otherwise non-zero. The values of the leftmost three bits of the returned value are currently unspecified. If a frame was obtained, the first five bytes of the frame are copied into the array specified by *name*. Frames with errors are discarded; a count is kept for each type of error. Frames may be discarded for any of the following reasons: (1) CRC error, (2) frame too short (less than four bytes), (3) frame too long (buffer size exceeded), or (4) no receive buffer available. If a frame with a buffer attached was previously obtained with **rcvfrm**, but the buffer has not been released to the driver with **rtnrfrm**, that buffer is returned to the queue of empty receive buffers. At most one receive frame with no buffer attached is retained by the interpreter; if a new frame arrives before the frame with no buffer attached has been obtained with **rcvfrm**, the new frame is discarded.

rtnrfrm()
: Return a receive buffer. The current receive buffer (the one obtained by the most recent **rcvfrm** primitive) is returned to the driver. If there is no current receive buffer, the process is terminated in error.

rsxmtq()
: Reset the transmit-buffer queue. The sequence number assignment is removed from all transmit buffers. If a transmission is currently in progress, the transmission is aborted, if possible.

rtnxfrm(*primary*)
: Return a transmit buffer. The transmit buffer currently attached to the sequence number specified by the value of the primary expression is returned to the driver and the sequence number assignment is removed from that buffer. If the specified sequence number does not have a buffer attached, the process is terminated in error. Transmit buffers must be returned in the same sequence in which they were obtained,

otherwise the process is terminated in error.

setctl(*name*,*primary*)
: Specify transmit-control information. The number of bytes specified by the primary expression are copied from the array specified by *name* and saved for use with subsequent **xmtfrm** or **xmtctl** primitives. If the transmitter is currently busy, the process is terminated in error.

xmtbusy( )
: Test for transmitter busy. If a frame is currently being transmitted, the returned value is **true** (non-zero); otherwise the returned value is **false** (zero).

xmtctl( )
: Transmit a control frame. If a transmission is not already in progress, a new transmission is initiated. The transmitted frame will contain the control information specified by the most recent **setctl** primitive, followed by a two-byte CRC. The CRC-CCITT polynomial ($x^{16}+x^{12}+x^5+1$) is used for the CRC calculation. The returned value is zero if a new transmission was initiated, otherwise non-zero.

xmtfrm(*primary*)
: Transmit an information frame. If a transmission is not already in progress, a new transmission is initiated. The transmitted frame will contain the control information specified by the most recent **setctl** primitive, followed by the contents of the buffer which is currently attached to the sequence number specified by the value of the primary expression followed by a two-byte CRC. The CRC-CCITT polynomial ($x^{16}+x^{12}+x^5+1$) is used for the CRC calculation. The returned value is zero if a new transmission was initiated, otherwise non-zero. If the sequence number is greater than seven or the sequence number does not have a buffer attached, the process is terminated in error.

The following primitives are available with all versions of the interpreter:

dsrwait( )
: Wait for modem-ready and then set modem-ready mode. The process delays until the modem-ready signal from the modem interface is asserted. If the modem-ready signal subsequently drops, the process is terminated. If **dsrwait** is never invoked, the modem-ready signal is ignored.

exit(*primary*)
: Terminate execution. The process is halted and the value of the primary expression is passed to the driver.

getcmd(*name*)
: Get a command from the driver. If a command has been received from the driver since the last call to **getcmd**, four bytes of command information are copied into the array specified by *name* and a value of **true** (non-zero) is returned. If no command is available, the returned value is **false** (zero).

pause( )
: Return control to the dispatcher. This primitive informs the dispatcher that the virtual process may be suspended until the next occurrence of an event that might affect the state of the protocol for this line. Examples of such events are: (1) completion of an output transfer, (2) completion of an input transfer, (3) timer expiration, and (4) a buffer-in command from the driver. In a multi-line implementation, the **pause** primitive allows the process for a given line to give up control to allow the processor to service another line. In a single-line implementation this primitive has no effect.

snap(*name*)
: Create a *snap* event record. Four bytes from the array specified by *name* are passed to

the driver, which prefixes a time stamp and sequence number and creates a trace event record containing the data. If minor device 1 of the trace driver is currently open, the record is placed on the read queue for that device; otherwise the event record is discarded. The information passed via the *snap* primitive can be displayed using the *vpmsnap* command (see *vpmstart*(1C)).

rtnrpt(*name*)
: Return a report to the driver. Four bytes from the array specified by *name* are transferred to the driver. The process delays until the transfer is complete.

testop(*primary*)
: Test for odd parity. The returned value is **true** (non-zero) if the value of the primary expression has odd parity, otherwise the returned value is **false** (zero).

timeout(*primary*)
: Schedule or cancel a timer interrupt. If the value of the primary expression is non-zero, the current values of the program counter and stack pointer are saved and a timer is loaded with the value of the primary expression. The system call then returns immediately with a value of **false** (zero) as the returned value. The timer is decremented each tenth of a second thereafter. If the timer is decremented to zero, the saved values of the program counter and stack pointer are restored and the system call returns with a value of **true** (non-zero). The effect of the timer interrupt is to return control to the code immediately following the **timeout** system call, at which point a non-zero return value indicates that the timer has expired. The **timeout** system call with a non-zero argument is normally written as the condition part of an **if** statement. A **timeout** system call with a zero argument value cancels all previous **timeout** requests, as does a **return** from the function in which the **timeout** system call was made. A **timeout** system call with a non-zero argument value overrides all previous **timeout** requests. The maximum permissible value for the argument is 255, which gives a timeout period of 25.5 seconds.

timer(*primary*)
: Start a timer or test for timer expiration. If the value of the primary expression is non-zero, a software timer is loaded with the value of the primary expression and a value of **true** (non-zero) is returned. The timer is decremented each tenth of a second thereafter until it reaches zero. If the value of the primary expression is zero, the returned value is the current value of the timer; this will be **true** (non-zero) if the value of the timer is currently non-zero, otherwise **false** (zero). The timer used by this primitive is different from the timer used by the **timeout** primitive.

trace(*primary*[,*primary*])
: The values of the two primary expressions and the current value of the script location counter are passed to the driver, which prefixes a sequence number and creates a trace event record containing the data. If minor device 0 of the trace driver is currently open, the record is placed on the read queue for that device; otherwise the event record is discarded. The information passed via the *trace* primitive can be displayed using the *vpmtrace* command (see *vpmstart*(1C)). If the second argument is omitted, a zero is used instead. The process delays until the values have been accepted by the host computer.

Constants

A *constant* is a decimal, octal, or hexadecimal integer, or a single character enclosed in single quotes. A token consisting of a string of digits is taken to be an octal integer if the first digit is a zero, otherwise the string is interpreted as a decimal integer. If a token begins with **0x** or **0X**, the remainder of the token is interpreted as a hexadecimal integer. The hexadecimal digits include **a** through **f** or, equivalently, **A** through **F**.

Variables

Variable names may be used without having been previously declared. All names are global. All values are treated as 8-bit unsigned integers.

Arrays of contiguous storage may be allocated using the **array** declaration:

array *name*[*constant*]

where *constant* is a decimal integer. Elements of arrays can be referenced using constant subscripts:

name[*constant*]

Indexing of arrays assumes that the first element has an index of zero.

Names

A *name* is a sequence of letters and digits; the first character must be a letter. Upper- and lower-case letters are considered to be distinct. Names longer than 31 characters are truncated to 31 characters. The underscore (_) may be used within a name to improve readability, but is discarded by RATFOR.

Preprocessor Commands

If the **−m** option is omitted, comments, macro definitions, and file inclusion statements are written as in C. Otherwise, the following rules apply:

1. If the character **#** appears in an input line, the remainder of the line is treated as a comment.

2. A statement of the form:
   define(*name*,*text*)
   causes every subsequent appearance of *name* to be replaced by *text*. The defining text includes everything after the comma up to the balancing right parenthesis; multi-line definitions are allowed. Macros may have arguments. Any occurrence of **$***n* within the replacement text for a macro will be replaced by the *n*th actual argument when the macro is invoked.

3. A statement of the form:
   include(*file*)
   inserts the contents of *file* in place of the **include** command. The contents of the included file is often a set of definitions.

EXAMPLES

These examples require the use of the **−m** option.

```
# The function defined below transmits a frame in transparent BISYNC.
# A transmit buffer must be obtained with getxbuf before the function
# is invoked.
#
# Define symbolic constants:
#
define(DLE,0x10)
define(ETB,0x26)
define(PAD,0xff)
define(STX,0x02)
define(SYNC,0x32)
#
# Define a macro with an argument:
#
define(xmtcrc,{crc16($1); xmt($1);})
```

```
#
# Declare an array:
#
array crc[2];
#
# Define the function:
#
function xmtblk()
        crcloc(crc);
        xsom(SYNC);
        xmt(DLE);
        xmt(STX);
        while(get(byte)==0){
                if(byte == DLE)
                        xmt(DLE);
                xmtcrc(byte);
        }
        xmt(DLE);
        xmtcrc(ETB);
        xmt(crc[0]);
        xmt(crc[1]);
        xeom(PAD);
end
#
# The following example illustrates the use of macros to simulate a
# function call with arguments.
#
# The macro definition:
#
define(xmtctl,{c=$1;d=$2;xmtctl1()})
#
# The function definition:
#
function xmtctl1()
        xsom(SYNC);
        xmt(c);
        if(d!=0)
                xmt(d);
        xeom(PAD);
end
#
# Sample invocation:
#
function test()
        xmtctl(DLE,0x70);
end
```

FILES

sas_temp*	temporaries
/tmp/sas_ta??	temporary
/tmp/sas_tb??	temporary
/usr/lib/vpm/pass*	compiler phases
/usr/lib/vpm/pl	compiler phase

/usr/lib/vpm/vratfor	compiler phase
/lib/cpp	preprocessor
/usr/bin/m4	preprocessor
/bin/kasb	KMC11-B assembler
/usr/lib/vpm/bisync/*	interpreter source for the BISYNC interpreter
/usr/lib/vpm/hdlc/*	interpreter source for the HDLC interpreter

SEE ALSO

m4(1), ratfor(1), vpmstart(1C), vpm(4).
C Reference Manual by D. M. Ritchie.
RATFOR−A Preprocessor for a Rational Fortran by B. W. Kernighan.
The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.
Software Tools by B. W. Kernighan and P. J. Plauger (pp. 28-30).

NAME

vpmsave, vpmsnap, vpmtrace, vpmfmt − save and print VPM event traces

SYNOPSIS

vpmsave mask device

vpmsnap mask

vpmtrace mask

vpmfmt

DESCRIPTION

Vpmsave opens the minor device of the trace driver specified by *device*, enables the channels specified by *mask* (octal), and then reads event records and writes them to its standard output (unformated) until killed.

Vpmtrace opens */dev/trace* (assumed to be minor device 0 of the trace driver) and enables the channels specified by *mask* (octal). It then reads event records and prints them until killed.

Vpmsnap opens */dev/snap* (assumed to be minor device 1 of the trace driver) and enables the channels specified by *mask* (octal). It then reads time-stamped event records and prints them until killed.

Vpmfmt reads its standard input, which it assumes was generated by *vpmsave*, and prints it formatted to its standard output.

Note that

```
vpmsave mask device | vpmfmt
```

is equivalent to

```
vpmtrace mask
```

where *device* is the name of minor device 0 of the trace driver. If *device* is the name of minor device 1 of the trace driver, it is almost equivalent to

```
vpmsnap mask
```

the only difference being that the times are not normalized to zero.

Vpmsave and *vpmfmt* are provided because fewer event records are lost when they are used as follows:

```
vpmsave mask device > t &

vpmfmt < t
```

SEE ALSO

vpmc(1C), kmc(4), trace(4), vpm(4).

NAME

vpmset, vpmstart − connect VPM drivers and KMCs; load the KMC11-B.

SYNOPSIS

vpmset pdev idev kdev [lineno]

vpmstart device n [filen]

DESCRIPTION

The *vpmset* command provides a means for dynamically associating a VPM protocol driver minor device with a particular KMC11 microcomputer or a particular line on a KMS11 communications mutiplexor. Each such connection requires the use of a separate VPM interface driver minor device as an intermediary. Until these connections have been made, a user program cannot open the VPM protocol minor device for reading and/or writing. These connections can be changed provided the VPM protocol minor device is not open for reading and/or writing and the VPM interface driver is not connected to a KMC or the protocol associated with the interface driver is not running (see the VPM interface functions *vpmstart* and *vpmstop* (vpm(4)).

Example:

```
vpmset /dev/vpm2 /dev/vpb3 /dev/kmc1 4
```

Vpmstart writes *filen* (**a.out** by default) to the KMC11-B specified by *device* and initiates execution.

The argument *n* is a magic number that the KMC driver saves to identify the running program. This number is checked when the VPM driver is opened to provide some assurance that the program running in the KMC is the one expected. The magic number for VPM interpreters is 6. When *filen* has been loaded into to the KMC, its execution is begun. *Filen* may be any file executable by the KMC.

If *filen* was made using *vpmc*(1C), the VPM interpreter will be started by *vpmstart*. The VPM interpreter waits for a RUN command from the VPM interface driver before beginning execution of the protocol script. The RUN command is sent by the VPM interface driver when the VPM protocol minor device is opened.

SEE ALSO

kmc(4), vpm(4).

NAME

vcrt − filter nroff output for virtual crts

SYNOPSIS

vcrt

DESCRIPTION

Vcrt reads from the standard input and writes onto the standard output. Its input is assumed to be from nroff running with the **−T43** option, and its output is intended for a crt using the virtual crt protocol. While *vcrt* will handle reverse paper motion correctly, if the crt being used does not have enough memory to hold at least a page of text, double column output will not work correctly unless the nroff output is first piped through *col*(1) using *col*'s **−f** option.

When outputting to a fully capable crt, *vcrt* will result in overstruck characters being in bold intensity, underlined characters underlined, greek characters in reverse video, superscripts in dim video, and subscripts in dim, underlined video.

SEE ALSO

nroff(1), col(1), over(1)

BUGS

Chinese characters come out sideways.

NAME
wait − await completion of process

SYNOPSIS
wait

DESCRIPTION
Wait until all processes started with & have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the Shell itself executes *wait,* without creating a new process.

SEE ALSO
sh(1)

BUGS
Not all the processes of a 3- or more-stage pipeline are children of the Shell, and thus can't be waited for.

NAME

wall − write to all users

SYNOPSIS

wall [−g grpname] [file]

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message to all currently logged in users preceded by 'Broadcast Message from *yourname* (ln??) ...'. It is used to warn all users, typically prior to shutting down the system.

If a file is given *wall* will read the file instead of the standard input.

The sender should be super-user to override any protections the users may have invoked.

If the -g option is used, a legitimate group name must be supplied. In this case the wall goes only to members of the group who are currently logged on to the system.

FILES

/dev/ln??

SEE ALSO

mesg(1), write(1)

DIAGNOSTICS

"Can't fork!"

"Cannot send to ..." when the open on a user's terminal file fails.

NAME

wc − word count

SYNOPSIS

wc [**−lwc**] [name ...]

DESCRIPTION

Wc counts lines, words and characters in the named files, or in the standard input if no name appears. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **−lwc**.

When file names are specified on the command entry line the name(s) will be printed along with any counts.

NAME

what − identify files

SYNOPSIS

what file

DESCRIPTION

What searches the given files for all occurrences of the pattern which *get*(1S) substitutes for %Z% (this is **@(#)** at this printing) and prints out what follows until the first ", >, newline, \, or null character. For example, if the C program in file **f.c** contains

```
char iden___[] "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

what f.c f.o a.out

will print

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

What is intended to be used in conjunction with the command *get*(1S), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

SEE ALSO

get(1S), help(1S), stamp(1)

DIAGNOSTICS

Use *help*(1S) for explanations.

BUGS

It's possible that an unintended occurrence of the pattern **@(#)** could be found just by chance, but this causes no harm in nearly all cases.

NAME

who − who is on the system

SYNOPSIS

who [**−rbtpluda**] [who-like file]

who [**am i**]

DESCRIPTION

Who, without an argument, lists the name, whether the user can write to the line, line id, login time, how long since output occurred to line, and the process id of the command interpreter(shell) for each current UNIX user. *Who* with the **am i** option identifies the user.

Without an argument, *who* examines the **/etc/utmp** file to obtain its information. If a file is given, that file is examined. Typically the given file will be **/etc/wtmp**, which contains a record of all the logins since it was created.

The general format for all entries is:

name [state] line time activity pid [loc|id] [exit]

With switches *who* will list logins, logouts, reboots, and modifications of the system clock, as well as other processes spawned by the init process.

−u This is the default option and lists only those users who are currently logged in. The *name* is the user's login name. The *state* describes whether someone else can write to that terminal. A '+' appears if the terminal is writable by anyone. A '−' appears if it is not. If the person executing the *who* command is **root**, then an '**x**' will appear for lines which have the *exclusive use* bit set and thus are not writable by **root**. All other lines will have a '+' or '−' as is appropriate. **Root** can write to all lines having a '+' or a '−'. If a bad line is encountered, a '**?**' is printed. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since output last went to that particular line. A '**.**' indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours elapse, the entry is marked '**old**'. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process id of the user's shell. The *loc* field is the comment field associated with this line as found in **/etc/inittab.** This is usually contains information about where the terminal is located.

−l This option lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field doesn't exist.

−p This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *activity* fields have no meaning. The *loc* field is replaced with the *id* field, which is the first two characters of the line in **/etc/inittab** that spawned this process.

−d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the exit status of the dead process. This can be useful in determining why a process terminated.

−b lists the last instance of a hardware boot successfully invoking the *init* process.

−r indicates the run state *init* has been placed in.

−t lists changes to the system clock and who made them.

−a lists all options.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), init(1M), login(1), mesg(1), inittab(5), utmp(5)

NAME

whodo − who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

Whodo produces merged, reformatted, and dated output from the *who*(1) and *ps*(1) commands.

NAME

write – write to another user

SYNOPSIS

write user [line]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message

Message from *yourname* (**ln**??)...

to the person you want to talk to. When it has successfully completed the connection it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point *write* writes 'EOT' on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or tty to send to; otherwise the first instance of the user found in the who (**/etc/utmp**) file is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal

Permission to write may be denied or granted by use of the *mesg* command. At the outset writing is allowed. Certain commands, in particular *nroff* and *pr* disallow messages in order to prevent messy output. However, if the user has super user permissions messages can be forced onto a write inhibited terminal.

If the character '!' is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first write to another user, wait for him or her to write back before starting to send. Each party should end each message with a distinctive signal ((**o**) for 'over' is conventional) that the other may reply. (**oo**) (for 'over and out') is suggested when conversation is about to be terminated.

FILES

/etc/utmp	to find user /bin/sh to execute '!'

SEE ALSO

mail(1), mesg(1), who(1)

DIAGNOSTICS

user not logged in

NAME

wump − hunt the wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

Wump plays the game of "Hunt the Wumpus." A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company, 2,* 2 (November 1973).

BUGS

It will never replace Space War.

NAME

x25pvc, x25lnk − install, remove, or get status for a PVC or BX.25 link

SYNOPSIS

x25pvc options

x25lnk options

DESCRIPTION

X25pvc may be used to install or remove a BX.25 Permanent Virtual Circuit (PVC) on a specified BX.25 interface (*link*), or to display the status of a specified BX.25 minor device (*slot*). Exactly one of the following options must be used:

−i [**−S**] [**−R**] [**−N**] *slotname chno linkno*

Slotname is a path name that specifies a BX.25 minor device (slot). If the minor device is currently connected to some logical channel on some BX.25 interface (link), then first that minor device will be removed, if possible (see the **−r** option). If that minor device is available, it is connected to logical channel *chno* on link number *linkno*. *Chno* must be in the range of 1 to 4,095 and must not be currently in use for any other BX.25 minor device associated with that link. Exactly one of three options must be used to indicate the session-establishment protocol to be used. The **−S** option, which is the preferred option, indicates that session-layer connect/accept/disconnect qualified data messages are to be used. The **−R** option indicates that RESET in-order/out-of-order packets will be recognized. The **−N** option indicates that the "no protocol" session mode is used.

−r *slotname*

Remove BX.25 minor device *slotname*. The command will fail if the slot is open, if packets are waiting to be transmitted, or if there are unacknowledged packets outstanding.

−s *slotname*

Print the status of BX.25 minor device *slotname*. The information printed consists of *slotname*, the logical channel number, the link number, and the session-establishment option.

X25lnk is used to activate or deactivate a specified BX.25 link. Exactly one of the following options must be used:

−i [**−b**] [**−p** *pktsize*] *linkno vpmb kmc lineno* [*kdm*]

Activate the BX.25 link that is specified by *linkno*. The **−b** option specifies that the link-level protocol will use Address B. The default is Address A. The **−p** option defines the packet size; if it is used, *pktsize* must be a number that is a power of 2 between 16 and 1,024 inclusive. The default packet size is 128. *Linkno* is the number of the BX.25 link to be installed. *Vpmb* is a path name that specifies the minor device number of the VPM interface driver that is to be used for the link. *Kmc* is a path name that specifies the minor device of the KMC driver and *lineno* specifies one of the eight lines (0-7) of a KMS11 multiple-line hardware interface. *Kdm* is a pathname that specifies the minor device of the DM11 modem control driver. The **−i** option makes the necessary connections between data structures and starts the BX.25 protocol on the link.

−h *linkno*

Halt the link specified by *linkno*.

−s *linkno*

Print the status of the link specified by *linkno*. The information printed consists of the link number, the packet size used on that link, the minor device number of the VPM interface driver, the minor device number of the KMC driver, and the line number of the KMC or KMS.

SEE ALSO

nc(4), vpm(4), x25(4).
Operations Systems Network Protocol Specification: BX.25 Issue 2, Bell Laboratories.

NAME

xargs − construct argument lists and execute command

SYNOPSIS

xargs [-*count*] command [initial arguments]

DESCRIPTION

Xargs concatenates the given command (and initial arguments) with lines from the standard input. The command is executed for every 20 lines of input, unless the optional *count* is given.

EXAMPLE

The command line

```
find . −type f −print | xargs −5 echo
```

will execute the *echo* command for each 5 lines delivered by *find* (i.e., printed 5 per output line).

NAME

xref − cross reference for C programs

SYNOPSIS

xref [file ...]

DESCRIPTION

xref reads the named *files* or the standard input if no file is specified and prints a cross reference consisting of lines of the form

```
          identifier          file-name          line-numbers ...
```

Function definition is indicated by a plus sign (+) preceding the line number.

SEE ALSO

cref(1)

NAME

yacc − yet another compiler-compiler

SYNOPSIS

yacc [**−vd**] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*.

If the **−v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **−d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned 'token codes' with the user-declared 'token names'. This allows source files other than **y.tab.c** to access the token codes.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(I)
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.
YACC − Yet Another Compiler Compiler by S. C. Johnson.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.