An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character ^ is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The ^ loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u

The *u*ndo command reverses the effect of the last *s* command. The *u* command affects only the last line changed by the most recent *s* command. Some commands will cause the last substitution to be forgotten and the *u*ndo command will not work.

(1,$)v/*RE*/*command list*

This command is the same as the global command *g* except that the *command list* is executed with . initially set to every line that does *not* match the RE.

(1,$)V/*RE*/

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1,$)w *name*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *name* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If the name used in a *w* command begins with !, the rest of the line is taken to be a shell (*sh*(1)) command to be written to. Such a command is *not* remembered as the current file name.

x

A key string is demanded from the standard input. Later *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

($)=

The line number of the addressed line is typed; . is unchanged by this command.

!*shell command*

The remainder of the line after the ! is sent to the UNIX shell (*sh*(1)) to be interpreted as a command. Within the text of the command, the character % is replaced with the current filename and ! is replaced with the text of the previous command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1)<new-line>
An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to .+1p; it is useful for stepping forward through the buffer.

The editor has a limited macro capability. Macros are defined at the beginning of *any* line, even in the append mode. The following is a correct macro definition:

\\C=anystring

Macro definitions are intercepted at the getchar() level of the editor and are recognized by the following sequence: <newline><backslash><backslash><uppercase_letter><=>. The "anystring" is any string, including one with escaped newlines. Thus, it could be a series of commands to the editor, including multiple append sequences. A macro is invoked whenever a string of the form <backslash><uppercase_letter> is seen. If the preceding sequence is preceded by a backslash, translation is turned off. Thus to get the sequence "\C" in to the editor when the "\C" macro is defined, type "\\C". Legal macro names are all upper case letters.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a ? and returns to its command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. Files (e.g., *a.out*) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

FILES

/tmp/e#	temporary; # is the process number.
ed.hup	work is saved here if the terminal is hung up.

DIAGNOSTICS

?	for command errors.
?*name*	for an inaccessible file.
?TMP	for overflow of temporary file.
	(use the *h*elp and *H*elp commands for more detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer into the remembered name, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *q* or *e* commands: it prints ? and allows one to continue editing. A second *q* or *e* command at this point will take effect. The − command-line option inhibits this feature.

SEE ALSO

crypt(1), grep(1), sed(1), sh(1).
A Tutorial Introduction to the UNIX Text Editor by B. W. Kernighan.
Advanced Editing on UNIX by B. W. Kernighan.

BUGS

Escaped new-lines do not work in the replacement string of a *s* command that is part of the *command list* of a *g* or a *v* command.
A *!* command cannot be subject to a *g* or a *v* command.
The sequence \n in a RE does not match any character.
The *l* command mishandles DEL.
Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.

NAME

env - set environment for command execution

SYNOPSIS

env [-] [name=value] ... [command args]

DESCRIPTION

Env obtains the current *environment,* modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The '-' flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO

sh(1), exec(2), environ(7)

NAME

epoch — print and set system backup date

SYNOPSIS

epoch [—] [mmddhhmm[yy]]

DESCRIPTION

Epoch with no arguments prints the most recent system backup date. With the optional '—', *epoch* saves the current date in the epoch file. An argument of the type *mmddhhmm*[*yy*], will be used to set the epoch date (see *date*(1)).

The epoch file, */etc/epoch*, contains 4 bytes (long int) which is the epoch date measured in seconds from 0000 GMT Jan 1 1970.

FILES

/etc/epoch epoch file

SEE ALSO

date(1), time(2)

NAME

eqn, neqn, checkeq − typeset mathematical text

SYNOPSIS

eqn [**−dxy**] [**−pn**] [**−sn**] [**−fn**] [file] ...
neqn [**−dxy**] [**−pn**] [**−sn**] [**−fn**] [file] ...
checkeq [file] ...

DESCRIPTION

Eqn is a *troff*(1) preprocessor for typesetting mathematical text on a Wang-Graphic Systems, Inc. phototypesetter, while *neqn* is used for the same purpose with *nroff*(1) on typewriter-like terminals. Usage is almost always:

```
eqn file ... | troff
neqn file ... | nroff
```

If no files are specified, these programs read from the standard input. A line beginning with .EQ marks the start of an equation; the end of an equation is marked by a line beginning with .EN. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as *delimiters*; subsequent text between delimiters is then treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument **−d***xy* or (more commonly) with **delim** *xy* between .EQ and .EN. The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by **delim off**. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within *eqn* are separated by spaces, tabs, new-lines, braces, double quotes, tildes, or circumflexes. Braces { } are used for grouping; generally speaking, anywhere a single character such as *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde ~ represents a full space in the output, circumflex ^ half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub j* makes x_j, *a sub k sup 2* produces a_k^2, and *e sup {x sup 2 + y sup 2}* gives $e^{x^2+y^2}$.

Fractions are made with **over**: *a over b* yields $\frac{a}{b}$.

sqrt makes square roots: *1 over sqrt {ax sup 2+bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim_{n\to\infty} \sum_{0}^{n} x_i$ is made with *lim from {n−> inf} sum from 0 to n x sub i*.

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: *left [x sup 2 + y sup 2 over alpha right] ~=~ 1* produces $\left[x^2+\frac{y^2}{\alpha}\right] = 1$. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile; **lpile** left-justifies, **pile** and **cpile** center (but with different vertical spacing), and **rpile** right justifies.

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x}=\overline{f(t)}$, *y dotdot bar ~ =~ n under* is $\ddot{y} = \underline{n}$, and *x vec ~ =~ y dyad* is $\vec{x} = \overline{\vec{y}}$.

Point sizes and fonts can be changed with **size** *n* or **size** ±*n*, **roman**, **italic**, **bold**, and **font** *n*. Point sizes and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **−s***n* and **−f***n*.

Normally, subscripts and superscripts are reduced by 3 points from the previous size; this may be changed by the command-line argument **−p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

define thing % replacement %

defines a new token called *thing* that will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords such as **sum** (∑), **int** (∫), **inf** (∞), and shorthands such as >= (≥), != (≠), and −> (→) are recognized. Greek letters are spelled out in the desired case, as in **alpha** or GAMMA. Mathematical words such as **sin**, **cos**, and **log** are made Roman automatically. *Troff*(1) four-character escapes such as \(**dd** (‡) and \(**bs** (⏎) may be used anywhere. Strings enclosed in double quotes " ... " are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff*(1) when all else fails. Full details are given in the manual cited below.

SEE ALSO

Typesetting Mathematics − User's Guide by B. W. Kernighan and L. L. Cherry
mm(1), mmt(1), tbl(1), troff(1), eqnchar(7), mm(7), mv(7).

BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in **bold "12.3"**.

NAME

errdead – extract error records from dump

SYNOPSIS

/etc/errdead [dumpfile] [namelist]

DESCRIPTION

When hardware errors are detected by the system, an error record that contains information pertinent to the error is generated. If the error-logging daemon *errdemon*(1M) is not active or if the system crashes before the record can be placed in the error file, the error information is held by the system in a local buffer. *Errdead* examines a system dump (or memory), extracts such error records, and passes them to *errpt*(1M) for analysis.

The *dumpfile* specifies the file (or memory) that is to be examined. The system namelist is specified by *namelist;* if not given, **/unix** is used. If the *dumpfile* is missing, **/dev/mem** is used.

FILES

/unix	system namelist
/dev/mem	current core image
/usr/bin/errpt	analysis program
/usr/tmp/errXXXXXX	temporary file

DIAGNOSTICS

Diagnostics may come from either *errdead* or *errpt*. In either case, they are intended to be self-explanatory.

SEE ALSO

errpt(1M), errdemon(1M)

NAME

errdemon − error-logging daemon

SYNOPSIS

/etc/errdemon [filename]

DESCRIPTION

The error logging daemon *errdemon* collects error records from the operating system by reading the special file **/dev/error** and places them in the file *filename*. If *filename* is not specified when the daemon is activated, the file **/errlog/errfile** is used. Note that *filename* is created if it does not exist; otherwise, error records are appended to it, so that no previous error data is lost. No analysis of the error records is done by *errdemon;* that responsibility is left to *errpt*(1M). The error-logging daemon is terminated by sending it a software kill signal (signal 15). Only the super-user may start the daemon, and only one daemon may be active at any time. Normally the daemon is started from an entry in the lines file.

FILES

/dev/error	source of error records
/errlog/errfile	repository for error records
/dev/systty	repository for error messages generated by the daemon.

DIAGNOSTICS

The diagnostics produced by *errdemon* are intended to be self-explanatory.

SEE ALSO

errpt (1M), kill(1), err(4), errdead(1), errfile(5)

NAME

errpt − process a report of logged errors

SYNOPSIS

errpt [**−a**] [*−dev*] ... [**−int**] [**−mem**] [**−power**] [**−ovfl**] [**−prdev**] - [**−s** *date*] [**−e** *date*] [**−p***n*] [**−f**] [file ...]

DESCRIPTION

Errpt processes data collected by the error logging mechanism (*errdemon*(1M)) and generates a report of that data. The default report is a summary of all errors posted in the files named. Options apply to all files and are described below. If no files are specified, *errpt* attempts to use **/errlog/errfile** as *file*.

A summary report notes the options that may limit its completeness, records the time stamped on the earliest and latest errors encountered, and gives the total number of errors of one or more types. Each device summary contains the total number of unrecovered errors, recovered errors, errors unabled to be logged, I/O operations on the device, and miscellaneous activities that occurred on the device. The number of times that *errpt* has difficulty reading input data is included as read errors.

Any detailed report contains, in addition to specific error information, all instances of the error logging process being started and stopped and any time changes (via *date*(1)) or configuration changes (for MERT environment only) that took place during the interval being processed. A summary of each error type included in the report is appended to a detailed report.

A report may be limited to certain records in the following ways:

−s *date* Ignore all records posted earlier than *date*, where *date* has the form **mmddhhmmyy**, consistent in meaning with the *date*(1) command.

−e *date* Ignore all records posted later than *date*, whose form is as described above.

−a Produce a detailed report that includes all error types.

−dev A detailed report is limited to *dev*, a block device identifier. *Errpt* is familiar with the common form of identifiers (e.g., rs03, RS04, hs; see Section 4 of this volume). Currently, the block devices for which errors are logged are RP03, RP04, RP05, RP06, RS03, RS04 TU10, TU16, RK05, RF11.

−int Include in a detailed report errors of the stray-interrupt type.

−mem Include in a detailed report errors of the memory-parity type.

−power Include in a detailed report any power-fail restarts.

−ovfl Include in a detailed report certain operating system table overflows. In particular report overflows of the file, process, inode and text tables.

−prdev Include in a detailed report filesystem errors. In particular report bad block, bad count, no space, and out of inode messages.

−p *n* Limit the size of a detailed report to *n* pages.

−f In a detailed report, limit the reporting of block device errors to unrecovered errors. For the *−ovfl* and *−prdev* options, the operating system will produce at most one message every 2 minutes if the same error is being encountered repetitively. Repetitions of the same message will be recorded as "unlogged" errors in a subsequent report. This technique is used to avoid excess operating system overhead when a resource becomes congested, unavailable, or full.

FILES

/errlog/errfile - default error file

SEE ALSO

errfile(5), errdemon(1M), errdead(1M)

NAME

expr — evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr \| *expr*
returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr \& *expr*
returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { =, \>, \>=, \<, \<=, != } *expr*
returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, − } *expr*
addition or subtraction of integer-valued arguments.

expr { *, /, % } *expr*
multiplication, division, or remainder of the integer-valued arguments.

expr : *expr*
The matching operator : compares the first argument with the second argument which must be a regular expression; regular expression syntax is the same as that of *ed*(1), except that all patterns are 'anchored' (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(... \) pattern symbols can be used to return a portion of the first argument.

ARCHAIC FORMS

The following operators are supported by the current version of **expr**, but have been made obsolete by the : operator. These should not be used in new applications.

substr *expra exprb exprc*
returns on standard output that portion of *expra* (possibily null) which is defined by the numerical offset (*exprb*, starting at **1**) and the numerical span (*exprc*). A large span value may be given to obtain the remainder of the string.
substr abcd 2 2 is equivalent to **abcd : '..\(..\)'**

length *expr*
returns the length in characters of the expression that follows.
length *expr* **is equivalent to** *expr* **: '.*'**

index *expra exprb*
searches the first expression for the first character that matches a character from the second expression. It returns the character position number if is succeds, or **0** if it fails.
index abcd d is equivalent to **abcd : d**

EXAMPLES

1. a=`expr $a + 1`

   adds 1 to the shell variable **a.**

2. : `For $a equal to either "/usr/abc/file" or just "file"`
   expr $a : `.*/\(.*\)` \| $a

   returns the last segment of a path name (i.e., file). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. : A better representation of example 2.
   expr //$a : `.*/\(.*\)`

   The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. expr $VAR : `.*`

   returns the number of characters in **$VAR.**

SEE ALSO

ed(1), sh(1).

EXIT CODE

As a side effect of expression evaluation, *expr* returns the following exit values:

0	if the expression is neither null nor **0**
1	if the expression *is* null or **0**
2	for invalid expressions.

DIAGNOSTICS

syntax error − for operator/operand errors
non-numeric argument − if arithmetic is attempted on such a string

BUGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an =, the command:

expr $a = `=`

looks like:

expr = = =

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

expr X$a = X=

NAME

f77 − FORTRAN 77 compiler

SYNOPSIS

f77 [option] ... file ...

DESCRIPTION

F77 is the UNIX Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with '.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with '.o' substituted for '.f'.

Arguments whose names end with '.r' or '.e' are taken to be Ratfor or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by f77.

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

The following options have the same meaning as in *cc*(1). See *ld*(1) for load-time options.

−c Suppress loading and produce '.o' files for each source file.

−p Prepare object files for profiling, see *prof*(1).

−O Invoke an object-code optimizer.

−S Compile the named programs, and leave the assembler-language output on corresponding files suffixed '.s'. (No '.o' is created.).

−o output
Name the final output file *output* instead of **a.out**.

−f Load programs with the floating point interpreter.

The following options are peculiar to *f77*.

−onetrip Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

−u Make the default type of a variable 'undefined' rather than using the default Fortran rules.

−w Suppress all warning messages. If the option is '−w66', only Fortran 66 compatibility warnings are suppressed.

−F Apply EFL and Ratfor preprocessor to relevant files, put the result in the file with the suffix changed to '.of', but do not compile.

−m Apply the M4 preprocessor to each EFL or Ratfor source file before transforming with the ratfor or efl processor.

−E The remaining characters in the argument are used as an EFL flag argument whenever processing a '.e' file.

−R The remaining characters in the argument are used as a Ratfor flag argument whenever processing a '.r' file.

Other arguments are taken to be either loader option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out**.

FILES

file.[fresc]	input file
file.o	object file

a.out	loaded output
./fort[pid].?	temporary
/usr/lib/f77pass1	compiler
/lib/c1	pass 2
/lib/c2	optional optimizer
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/lib/libc.a	C library; see Section 3 of this volume.

SEE ALSO

S. I. Feldman, P. J. Weinberger, *A Portable Fortran 77 Compiler (TM-78-1273-1 = TM-78-3444-1)*
prof(1), cc(1), ld(1)

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

BUGS

This program is too big to run on processors with no Floating Point hardware.

NAME

factor, primes – factor a number, generate large primes

SYNOPSIS

factor [number]

primes

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{56} (about 7.2×10^{16}) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to $\sqrt{n}$ and occurs when n is prime or the square of a prime. It takes 1 minute to factor a prime near 10^{14} on a PDP11.

When *primes* is invoked, it waits for a number to be typed in. If you type in a positive number less than 2^{56} it will print all primes greater than or equal to this number.

DIAGNOSTICS

'Ouch.' for input out of range or for garbage input.

NAME

fed — edit associative memory for form letter

SYNOPSIS

fed

DESCRIPTION

Fed is used to edit a form letter associative memory file, **form.m**, which consists of named strings. Commands consist of single letters followed by a list of string names separated by a single space and ending with a new line. The conventions of the Shell with respect to '*' and '?' hold for all commands but **m**. The commands are:

e name ...

Fed writes the string whose name is *name* onto a temporary file and executes *ed*. On exit from the *ed* the temporary file is copied back into the associative memory. Each argument is operated on separately. Be sure to give an *ed* **w** command (without a filename) to rewrite *fed's* temporary file before quitting out of *ed*.

d [name ...]

deletes a string and its name from the memory. When called with no arguments **d** operates in a verbose mode typing each string name and deleting only if a **y** is typed. A **q** response returns to *fed*'s command level. Any other response does nothing.

m name1 name2 ...

(move) changes the name of *name1* to *name2* and removes previous string *name2* if one exists. Several pairs of arguments may be given. Literal strings are expected for the names.

n [name ...]

(names) lists the string names in the memory. If called with the optional arguments, it just lists those requested.

p name ...

prints the contents of the strings with names given by the arguments.

q

returns to the system.

c [**p**] [**f**]

checks the associative memory file for consistency and reports the number of free headers and blocks. The optional arguments do the following:

p causes any unaccounted-for string to be printed.

f fixes broken memories by adding unaccounted-for headers to free storage and removing references to released headers from associative memory.

FILES

/tmp/ftmp?	temporary
form.m	associative memory

SEE ALSO

form(1), ed(1), sh(1)

WARNING

It is legal but unwise to have string names with blanks, ':' or '?' in them.

NAME

festoon − turgid memorandum composition

SYNOPSIS

/usr/games/festoon number-of-sentences [percent-invented-nouns]

DESCRIPTION

Because the abilities are being anniated by the conclusions, an undue number of requirements analysises that quadrimittated significantnesses were being ridized by a couple potential usefulnesses. Being as motities which are malpathating some reasonable compromises are gynening a necessaryness by confirmations, the implementation had been being associatively licitesced by necessary revisions. There is a redefinition that totally needfully trarogatesces this mention.

It is interesting to note that that criterion being caused by these realities may have instantly quadhydroened the conception. As is often the case, there is a strong feeling which had multcepened a clean interfacing to a good relationship. These prioritized total effects are peritestated by the well defined interfacing.

In the light of the fact that impediments liveren many design issues, a number of negative impacts were fectesced at a viewpoint. The methodological comprehensive plans are being euesced by that motivation. Inasmuch as the unprecedented terminology differently ennealucesced a transacence, a signification had been strikingly offgraphfied by a misostress in conflict with an interfacing.

With this in mind, those inclusions are antezoesced by a reification exhibiting a tendency towards a transitioning. Being as a capability in back of a policy in the field of effectations was misoated by a measurement in conjunction with a shortage, a standardized clear characteristicness may be fidfied by the tight schedule pressure. An agreement of these exemptions had been being antefactized by functional overviews. This replacement impacting a requirements definition could be being verized by a team responsibility.

Seen in the above light, the experience levels pelesced a very promptly modularized activation in conjunction with not unsufficient experience levels. A progress fundamentally steadily motened a cross attendance. If well defined interfacings were very lastly novemjurfying an essential essence, an evolving organization was pelably archating the complete revision. Needless to say, a study activity should principally eferate the expectations.

As regards the fact that the fact that the primary purpose was being continuously postmotfied by a very invaluably not unorganizing matter had been dependably jurfied by the order of magnitude, that discussion officially multiphobefied a functional overview. In light of these facts, this proposed enhancement needfully cidesced a quantitative result.

The full utilization can unliveresce some successes.

BUGS

An output that prepopizes this advent was misesced by the construction, yet a current proposal had been hectofacened by this underlying purpose. A few impediments have been viviened by this multiscripant. Those mechanizations will be sequfied by differences.

NAME

file − determine file type

SYNOPSIS

file [−**f**] file ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language.

If the −**f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

NAME

file_log − log an input string in a logfile.

SYNOPSIS

file_log input_string

DESCRIPTION

File_log will maintain an already existing file named by the environment variable **$DELTALOG**. When invoked with a string, it will lock the delta_log file (prevent other file_log invocations from accessing it) and append the string along with some identifying information. It then unlocks the file and exits. If an error occurs the entire log entry is mail(1)'ed to an administrator defined in the file_log shell file.

File_log uses a shell variable **$DELTALOG** to identify the log file. If it is not defineed or null it is assumed to be **delta_log**. The **$DELTALOG** environment variable is a relative path name of a file used for recording changes to the SCCS data base (i.e. source files). It searches each component of the environment variable **$SCCSOURCE** for a readable file named by **$DELTALOG**. Thus, each different **$SCCSOURCE** can have a different delta log file. *File_log* is written in Bourne shell-ese and thus runs only on systems supporting the Bourne shell.

File_log is used by gdelta(1S) and gadd(1S) to record all changes made to particular subsystems. If the file named by **$DELTALOG** does not exist, *file_log* just exits without logging anything.

As an example assume the SCCSOURCE and DELTALOG variables are set to:

```
SCCSOURCE=/usr/src
DELTALOG=admin/sys_dellog
```

Then,

```
gdelta -y'Added da11 driver' conf.c
```

causes the following information to be appended to the file /usr/src/admin/sys_dellog:

```
egb May 3 13:42:57
        /usr/source/src/ucb/os/s.conf.c 2.13
        -yAdded da11 driver
```

Note the "-y" is left in the comment. This is a simple way to distinguish between delta's and admin's (changes to the SCCS files as opposed to additions). *File_log* can be also be called directly.

SEE ALSO

gdelta(1S), gadd(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2.

NAME

find − find files

SYNOPSIS

find pathname expression

DESCRIPTION

Find recursively descends the directory hierarchy from *pathname* seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where +*n* means more than *n*, −*n* means less than *n* and *n* means exactly *n*.

−**ignore** *pathname*	True unless file is hierarchically below the given *pathname*. For maximum efficiency all *ignore* primaries should appear first in the boolean expression. In this case the files below the *pathname* will not be accessed.
−**name** *filename*	True if the *filename* argument matches the current file name. Normal *Shell* argument syntax may be used if escaped (watch out for **[**, **?**, and ***.**
−**perm** *onum*	True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags will be compared: *(flags&onum) == onum.*
−**i** *n*	True if the file has inode *n.*
−**type** *c*	True if the file is *c* , where *c* is **b**, **c**, **d**, or **f** for block special file, character special file, directory or ;lain file.
−**links** *n*	True if the file has *n* links.
−**user** *uname*	True if the file belongs to the user *uname*. An integer may be supplied instead of *uname.*
−**group** *gname*	True if the file belongs to the group *gname*. An integer may be supplied instead of *gname.*
−**size** *n*	True if the file is *n* blocks long (512 bytes per block).
−**atime** *n*	True if the file has been accessed in *n* days.
−**mtime** *n*	True if the file has been modified in *n* days.
−**exec** *command*	True if the executed command returns exit status zero (most commands do). The end of the command is punctuated by an escaped semicolon. A command argument '{}' is replaced by the current pathname.
−**ok** *command*	Like −**exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds **y**.
−**print**	Always true; causes the current pathname to be printed.

The primaries may be combined with the operators (ordered by precedence):

!	prefix *not*
−**a**	infix *and* second operand evaluated only if first is true
−**o**	infix *or,* second operand evaluated only if first is false
(*expression*)	parentheses for grouping. (Must be escaped.)

To remove files named 'a.out' and '*.o' not accessed for a week:

```
find / '(' −name a.out −o −name '*.o' ')' −a −atime +7 −a −exec rm {} ';'
```

FILES

/etc/group
/etc/passwd

SEE ALSO

sh(1), fs(5)

BUGS

There is no way to check device type.

NAME

flog − speed up a process

SYNOPSIS

flog [−**ln**] [−**am**] [−**u**] process-id

DESCRIPTION

Flog is used to stimulate an improvement in the performance of a process that has already been scheduled for execution by *spool*(1) or *wp*(1) commands. The *process-id* is the process number that is to be disiplined. The value *n* of the **l** keyletter argument is the flagellation constant, i.e. the number of *lashes* to be administered per minute. If this argument is omitted, the default is 17, which is the most random random number. The value *m* of the **a** keyletter argument is the number of times the inducement to speed up is to be *administered*. If this argument is ommitted, the default is one, which is based on the possibility that after *that* the process will rectify its behavior of its own volition. The presence of the **u** keyletter argument indicates that *flog* is to be *unmerciful* in its actions. This nullifies the effects of the other keyletter arguments. It is recommended that this option be used only on extremely stubborn processes, as its over-use may have detrimental effects.

FILES

Flog will read the file */have/mercy* for any entry containing the process-id of the process being speeded-up. The file can contain whatever supplications are deemed necessary, but, of course, these will be totally ignored if the **u** keyletter argument is supplied.

SEE ALSO

On Improving Process Performance by the Administration of Corrective Stimulation, *CACM*, vol. 4, 1657, pp. 356-654.
can(1), last(1), wp(1), spool(1), ops(1), backlog(1)

DIAGNOSTICS

If a named process does not exist, *flog* replies "flog you" on standard output. If *flog kill*(2)s the process, which usually happens when the **u** keyletter argument is supplied, it writes "rip," followed by the process-id of the deceased, on standard output.

BUGS

Spurious supplications for mercy by the process being flogged sometimes wind up on the standard output, rather that in */shut/up*.

WARNING − using *flog* more than once on any given job may cause the job to never be processd at all. The use of *flog* should be kept to a minimum!

If a job is *flogged* there is no way to "un-flog" it. Perhaps there is a need for an "apoligize" command?

NOTE:

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

NAME

form – form letter generator

SYNOPSIS

form proto arg ...

DESCRIPTION

Form generates a form letter from a prototype letter, an associative memory, arguments and in a special case, the current date.

If *form* is invoked with the *proto* argument *x*, the associative memory is searched for an entry with name *x* and the contents filed under that name are used as the prototype. If the search fails, the message '[*x*]:' is typed on the console and whatever text is typed in from the console, terminated by two new lines, is used as the prototype. If the prototype argument is missing, '**{letter}**' is assumed.

Basically, *form* is a copy process from the prototype to the output file. If an element of the form [*n*] (where *n* is a digit from 1 to 9) is encountered, the *n*-th argument is inserted in its place, and that argument is then rescanned. If [0] is encountered, the current date is inserted. If the desired argument has not been given, a message of the form '[*n*]:' is typed. The response typed in then is used for that argument.

If an element of the form [*name*] or {*name*} is encountered, the *name* is looked up in the associative memory. If it is found, the contents of the memory under this *name* replaces the original element (again rescanned). If the *name* is not found, a message of the form '[*name*]:' is typed. The response typed in is used for that element. The response is entered in the memory under the name if the name is enclosed in [ ]. The response is not entered in the memory but is remembered for the duration of the letter if the name is enclosed in {}.

In both of the above cases, the response is typed in by entering arbitrary text terminated by two new lines. Only the first of the two new lines is passed with the text.

If one of the special characters [{]}\ is preceded by a \, it loses its special meaning.

If a file named **forma** already exists in the user's directory, **formb** is used as the output file and so forth to **formz**.

The file **form.m** is created if none exists. Because **form.m** is operated on by the disc allocator, it should only be changed by using *fed*, the form letter editor, or *form*.

FILES

form.m associative memory
form? output file (read only)

SEE ALSO

fed(1), type(1), nroff(1)

BUGS

An unbalanced] or } acts as an end of file but may add a few strange entries to the associative memory.

NAME

g_find — locate and identify a source file

SYNOPSIS

g_find file

DESCRIPTION

The *g_find* command will locate a file within a directory structure defined by two global shell variables: **$SCCSOURCE** and **$SUBSYSTEMS**. *G_find* is written in Bourne shell-ese and thus runs only on systems supporting the Bourne shell. *G_find* requires two shell variables to be set and either made *global* or *exported*. The first is *SCCSOURCE*. It is set to be the directory which subtends all SCCS directories of current interest. Thus for the unix source software, *SCCSOURCE* is set to "/usr/src/ucb". The second variable is *SUBSYSTEMS*. It is set to the subdirectories of interest in $SCCSOURCE. Thus someone working on the operating system might set the following:

```
SCCSOURCE=/usr/src/ucb
SUBSYSTEMS="os io sys"
```

G_find will report back one of the following on the standard output:

a. file_name FILE

b. directory_name DIRECTORY

c. ERROR

The file_name output is the full pathname of the file starting with "/" and "s." prepended to the last component of the filename.

The directory_name output is a readable directory full path name.

The ERROR output indicates the appropriate SCCS file cannot be found in the $SCCSOURCE, $SUBSYSTEMS directory structure.

As an example assume the shell variables are set as above.

```
g_find os
```

causes the following output:

```
/usr/src/ucb/os DIRECTORY
```

G_find is used by gget(1S), gdelta(1S), gdiff(1S), gls(1S) and gprt(1S).

SEE ALSO

gget(1S), gdelta(1S), gprt(1S), gls(1S), gdiff(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2.

NAME

gadd − add a file to SCCS

SYNOPSIS

gadd [**−K**] [opts] subdir file [files]

DESCRIPTION

Gadd creates new SCCS files in the directory "$SCCSOURCE/subdir". It automatically inserts SCCS keywords identification strings unless the '**−K**' option is specified. All other options are assumed to be *admin*(1S) options and are passed accordingly. If successful, *gadd* leaves the named file in mode 0444. *Gadd* is written in Bourne shell-ese and thus runs only on systems supporting the Bourne shell. *Gadd* requires one shell variable to be set and either made *global* or *export*ed. $SCCSOURCE is set to be the directory which subtends all SCCS directories of current interest. Thus for the unix source software, *SCCSOURCE* is set to '**/usr/src/ucb**'. Thus someone working on the operating system might set the following:

SCCSOURCE=/usr/src/ucb

The remaining command line arguments are files. As an example assume the shell variable is set as above. Then

gadd os space.c

causes the file '/usr/src/ucb/os/s.space.c' to be created (as a read only file) whose inital contents are the file 'space.c'.

OPTIONS

−K suppress automatic addition of SCCS keywords.

SEE ALSO

admin(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2.

NAME

gadmin - admin a file in SCCS

SYNOPSIS

gadmin [options] [files]

DESCRIPTION

The *gadmin* command will do the *admin*(1S) command on the files named. All input flags are passed directly to the *admin (1S)* command. The files are accessed through the *g_find*(1S) command.

The command line arguments are files or any valid option to the *admin*(1S) command. If a *file* is specified, all $SUBSYSTEMS in $SCCSOURCE are searched in the order of $SUBSYSTEMS for a file named **s.***file*. When found, *gdelta* executes *admin*(1S) on the file with the appropriate directories and s. prepended to the file name.

FILES

/usr/bin/admin
/usr/sccsbin/g_find

SEE ALSO

admin(1S), g_find(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2, and are hopefully self explanatory.

NAME

gcat − send phototypesetter output to the HONEYWELL 6000

SYNOPSIS

gcat [option ...] [filename ...]

DESCRIPTION

Gcat arranges to have *troff*(1) output sent to the phototypesetter or debugging devices (STARE or line printer) attached to the HONEYWELL system. GCOS identification must appear in the UNIX password file (see *passwd*(5)). If no filename appears, the standard input is sent; thus *gcat* may be used as an output pipe for *troff*(1).

On many systems gcat is implemented as a shell file using *uux*(1). In this case the user must supply the normally optional **−i** option and the **−m** option will have no effect, though the user will receive remote mail when *gcat* is executed on the remote system.

The option **−g** (for GCOS) must be used with the *troff*(1) command to make things work properly. This command string sends output to the GCOS phototypesetter:

troff −g file | gcat

The following options, each as a separate argument, and in any combination (multiple outputs are permitted), may be given after gcat:

−ph Send output to the phototypesetter.
−st Send output to STARE for fast turn-around.
−tx Send output as text to the line printer (useful for checking spelling, hyphenation, pagination, etc.).
−du Send output to the line printer, dummied up to make the format correct. Because many characters are dropped, the output is unreadable, but useful for seeing the shape (margins, etc.) of the document.
−s*n* Submit job to GCOS with service grade *n* (*n*=1, 2, 3). Default is **−s1**.
−i Supply the GCOS "ident card" image as the parameter **−iMxxxx,Myyy** where **Mxxxx** is the GCOS job number and **Myyy** the GCOS bin number.
−c Make a copy of the file to be sent before returning to the user.
−r Remove the file after sending it.
−m When transmission is complete, report by *mail*(1) the so-called *snumb* of the receiving GCOS job. The mail is sent by the UNIX daemon; there is no guarantee that the GCOS job ran successfully. This is the default option.
−n Do not report the completion of transmission by *mail*(1).
−f Use the next argument as a dummy file name to report back in the mail. (This is useful for distinguishing multiple runs, especially when *gcat* is being used as a filter).
−o Print the on-line GCOS accounting output.
−t Toss out the on-line GCOS accounting output. This is the default option.

If none of the output options are specified, phototypesetter output (**−ph**) is assumed by default.

EXAMPLE

The command:

troff −g myfile | gcat −st −im1234,m567,myname −f myfile

will send the output of *troff*(1) to STARE, with the GCOS "ident card" specifying "M1234,M567,MYNAME", and will report back that **myfile** has been sent.

FILES

/usr/spool/dpd/*	spool area.
/etc/passwd	user's identification and accounting data.
/usr/lib/dpd	daemon.

SEE ALSO
uucp(1C), uux(1C)

NAME

gcon – convert GEX file to HIS format

SYNOPSIS

gcon [**–clxs***N*] [**file** ...]

DESCRIPTION

Gcon converts *gex*(1G) files into the format required by the *HIS* plotting routines. Output is to a file with the same name as the input file but suffixed with '**.his**'.

Options:

- **–c** Concatenate files. For example, file1 **–cc file2 file3**, will concatenate file2 and file3.
- **–l** Print graphic limits (extent). Turns off all other options.
- **–s***N* Set scale for plot.
- **–z** Adjust all points so that low x and low y are 0,0.

SEE ALSO

gex(1G)

AUTHOR

D. J. Jackowski

BUGS

Probably !
This program is under development and subject to change.

NAME

gdelta − delta a file from SCCS

SYNOPSIS

gdelta [options] [files]

DESCRIPTION

The *gdelta* command will *delta*(1S) the named source files into a file system dedicated to Source Code Control System (SCCS) files. *Gdelta* is written in Bourne shell-ese and thus runs only on systems supporting the Bourne shell. *Gdelta* requires two shell variables to be set and either made *global* or *export*ed. The first is *SCCSOURCE*. It is set to be the directory which subtends all SCCS directories of interest. Thus for the unix source software, *SCCSOURCE* is set to /usr/src/ucb . (On the Columbus system currently Unix source is under **/usr/src/ucb** in the directories **os**, **io**, and **sys**.) The second variable is *SUBSYSTEMS*. It is set to the subdirectories of interest in $SCCSOURCE. Thus someone working on the operating system might set the following:

SCCSOURCE=/usr/src/ucb
SUBSYSTEMS="os io sys"

The command line arguments are files or any valid option to the *delta*(1S) command. If a *file* is specified, all $SUBSYSTEMS in $SCCSOURCE are searched in the order of $SUBSYSTEMS for a file named **s.***file*. When found, *gdelta* executes *delta*(1S) on the file with the appropriate directories and '**s.**' prepended to the file name.

As an example assume the shell variables are set as above.

gdelta main.c

causes the following commands to be executed:

/usr/bin/delta $opts "$CMT" /usr/src/ucb/os/s.main.c

Here, **$opts** is a collection of all the input options except for the **−y** option to *delta*(1S). **$CMT** is the **−y** option if it exists. If the user does not use the **−y option,** *gdelta* prompts for the comment and allows multiline input. The input to the comment is terminated by an empty input line.

Gdelta also supports a logging mechanism. The *file_log*(1S) program is called to log the user, date, history, and the output from the *delta*(1S) command in the file determined to be the *delta* logging file. See *file_log*(1S).

FILES

/usr/bin/delta
/usr/sccsbin/dellog
/usr/sccsbin/file_log

SEE ALSO

dellog(1S), delta(1S), file_log(1S), gget(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2, and are hopefully self explanatory.

NAME

gdiff − diff an SCCS file with named file

SYNOPSIS

gdiff [**−rNN**] file [[**−rN**] file ...] ...

DESCRIPTION

The *gdiff* command will *gget*(1S) a file from SCCS directories and *diff*(1) the gget'ed file with the named file. See *gget*(1S) for which files are gget'ed. *Gdiff* is written in Bourne shell-ese and thus runs only on systems supporting the Bourne shell. *Gdiff* requires two shell variables to be set and either made *global* or *export*ed. The first is *SCCSOURCE*. It is set to be the directory which heads all SCCS directories of current interest. Thus for the unix source software, *SCCSOURCE* is set to /usr/src/ucb . The second variable is *SUBSYSTEMS*. It is set to the subdirectories of interest in $SCCSOURCE. Thus someone working on the operating system might set the following:

SCCSOURCE=/usr/src/ucb
SUBSYSTEMS="os io sys"

Gdiff will report back the differences between each named file and its corresponding SCCS file. As an example assume the shell variables are set as above.

gdiff tty.c

causes the following output:

/usr/src/ucb/s.tty.c -> tty.c
(*diff*(1) output)

Gdiff is essentially the following shell script:

gget -p tty.c | diff - tty.c

OPTIONS

−r Request a particular SID from the SCCS file. It is passed unscathed to the *gget*(1S) command.

SEE ALSO

gget(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2. Also occasional mysterious jackpots!

NAME

gdump - prints a gex graphic file

SYNOPSIS

gdump [file ...]

DESCRIPTION

Gdump prints a *gex*(1G) file in the format:

x, y, type, line style, [rotation/radius], [size], [symbol number], [text string]

Where *type* may be:

J - jump
V - vector
T - text
R - arc
C - circle
S - symbol

Example:

```
Wed Aug 24 13:51:40 1977    tx.g
15800 17600  J LS=0, 0
17000 17600  V LS=0, 0
17000 18100  V LS=0, 0
15800 18100  V LS=0, 0
15800 17600  V LS=0, 0
16100 16700  T LS=0, 0 ROT=0 SZ=200 A TEXT STRING
16100 16000  J LS=1, 1
18000 16000  V LS=1, 1
16800 15300  S LS=0, 0 ROT=0 SZ=200 SYM=100
15900 13300  J LS=0, 0
17100 13300  V LS=0, 0
18200 12700  R LS=0, 0
17100 12000  R LS=0, 0
15900 12000  V LS=0, 0
19700 12900  C LS=0, 0 RAD=70
```

SEE ALSO

gex(1G)

AUTHOR

D. J. Jackowski

NAME

get − get a version of an SCCS file

SYNOPSIS

get [−rSID] [−ccutoff] [−e] [−b] [−ilist] [−xlist] [−k] [−l[p]] [−p] [−s] [−m] [−n] [−g] [−t] [−aseq-no.] [−RSID] [−Mfilename] [−T] [−D] [−G] files

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with −. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

−r*SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the −e keyletter is also used), as a function of the SID specified.

−c*cutoff* *Cutoff* date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **−c7502** is equivalent to **−c750228235959**. Any number of non-numeric characters may separate the various 2 digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: "**−c77/2/2 9:22:25**". Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

~!get "−c%E% %U%" s.file

−e Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The −e keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *get*s for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **get** −e for different SIDs is always allowed.

If the *g-file* generated by *get* with an −e keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the −k keyletter in place of the −e keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the −e keyletter is used.

−b Used with the −e keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

−i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= SID | SID − SID

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

−x*list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **−i** keyletter for the *list* format.

−k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **−k** keyletter is implied by the **−e** keyletter.

−l[p] Causes a delta summary to be written into an *l-file*. If **−lp** is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.

−p Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **−s** keyletter is used, in which case it disappears.

−s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

−m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

−n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **−m** and **−n** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **−m** keyletter generated format.

−g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

−t Used to access the most recently created ("top") delta in a given release (e.g., **−r1**), or release and level (e.g., **−r1.2**).

−a*seq-no.* The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(5)). This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter, and users should not use it. If both the **−r** and **−a** keyletters are specified, the **−a** keyletter is used. Care should be taken when using the **−a** keyletter in conjunction with the **−e** keyletter, as the SID of the delta to be created may not be what one expects. The **−r** keyletter can be used with the **−a** and **−e** keyletters to control the naming of the SID of the delta to be created.

−R*SID* The SID-filename pairs in a file called *markfile* are retrieved. The *markfile* has the format of

<white_space>*filename*<white_space>SID

The effect of using the **−R** is

get −rSID s.markfile | get −

If no SID is specified, the highest level of the highest release of the *markfile* is used.

−M*filename*

Use *filename* instead of *markfile* when processing arguments to the **−R** option. If the **−R** flag is not specified, this option has no effect.

−T Causes the most recently created "top" delta of the *markfile* to be used when processing under the **−R** mode. The option has the same effect on the argument of **−R** as the **−t** has on the **−r** option. If the **−R** flag is not specified, this option has no effect.

−D Causes any directory structure under *file* (the filename arguments to *get*) to be recursively descended; the corresponding directories to be made, starting at the "." directory; and the files under each of the subdirectories under *file* to be retrieved. Thus, if

get −D /usr/src/lib

is typed and the user is in /usr/tmp, the entire directory structure and all clear text source files under /usr/src/lib would be reproduced under the /usr/tmp directory.

−G Causes each *markfile* to be read and printed. If the **−D** option is specified, a configuration listing will be produced. (I'm not sure about how well this works, I have not tested it fully {egb}.) If the **−R** flag is not specified, this option has no effect.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **−e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **−i** keyletter is used included deltas are listed following the notation "Included"; if the **−x** keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	**−b** Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	—	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	—	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	—	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1

R.L.B.S	−	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

\# Successor.

† The **−b** keyletter is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of − means "irrelevant".

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file (see *admin*(1)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see *admin*(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as "this shouldn't have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string @(#) recognizable by *what*(1).
%W%	A shorthand notation for constructing *what*(1) strings for UNIX program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing *what*(1) strings for non-UNIX program files. %A% = %Z%%Y% %M% %I%%Z%

FILES

Several auxiliary files may be created by *get*, These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading s with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For

example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **−p** keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **−k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **−l** keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
* otherwise.

b. A blank character if the delta was applied or wasn't applied and ignored;
* if the delta wasn't applied and wasn't ignored.

c. A code indicating a "special" reason why the delta was or was not applied:
"I": Included.
"X": Excluded.
"C": Cut off (by a **−c** keyletter).

d. Blank.

e. SCCS identification (SID).

f. Tab character.

g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.

h. Blank.

i. Login name of person who created *delta*.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an **−e** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **−e** keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **−i** keyletter argument if it was present, followed by a blank and the **−x** keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory

containing the SCCS files, but the real user doesn't, then only one file may be named when the −e keyletter is used.

NAME

getpc − get Program Counter data on running processes

SYNOPSIS

getpc [[**−k**] [**−u** pid ...] [**−g** pgrp] | [**−a**]] [**−t** secs]

DESCRIPTION

Getpc reads the special character device 'pcs' to retrieve Program Counter (pc) data for analysis by the *pcstat* command. The output from *getpc* is binary and put on its standard output. *Getpc* starts by producing the starting time and an entry identifying the type of profiling. At the end of profiling, the stop time is added to the output. *Getpc* also collects a summary of unasked-for data, which is produced by the system every 100 clock cycles. This extra data gives a summary of %mkerneli, %idle, etc. in the final report. At least one option must be specified, indicating the type of profiling desired. The options to *getpc* have the following meanings:

−k collect pcs from the UNIX kernel.

−u collect pcs from a list of user processes. The list is a list of process ids (maximum of 5 ids). Data is collected for each pid from both kernel and user mode. This option cannot be used in conjunction with the **−g** option!

−g collect pcs from the process group whose process group number is the operand of **−g**. Kernel data for the group is automatically gathered. This option cannot be used in conjunction with the **−u** option!

−a collect pcs from all user-level processes and the kernel. Specifying either the **−k**, **−u**, or **−g** option with **−a** is redundant and produces an error.

−t sets a time limit for pc collection (in seconds). If this option is not specified, *getpc* will run until killed.

Without any of **−k**, **−u**, **−g** or **−a** options, *getpc* gives a usage message. *getpc* writes the collected data to the standard output.

DIAGNOSTICS

getpc complains if it cannot open **/dev/pcs**.

SEE ALSO

pcstat(1), pcs(4)

NAME

getty − set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**−h**] line [speed [type [linedisc]]]
/etc/getty **−t** gettydefs-like-file

DESCRIPTION

Getty is a program that is invoked by *init*(1). It is the second process in the series, *init-getty-login-shell*, that ultimately connects a terminal user with UNIX. Initially *getty* generates a system identification message from the values returned by the *uname*(2) system call. Then, if **/etc/issue** exists, it outputs this to the user terminal, followed finally by the login message field for the entry it is using from **/etc/gettydefs**. *Getty* reads the user's login name and invokes the *login*(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used.

Line is the name of a tty line in '**/dev**' to which *getty* is to attach itself. *Getty* uses this string as the name of a file in the '**/dev**' directory to open for reading and writing. Unless *getty* is invoked with the **−h** flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The optional second argument, *speed,* is a label to a speed and tty definition in the file **/etc/gettydefs**(5). This definition tells *getty* what speed to initially run at, what the login message should look like, what the inital tty settings are, and what speed to try next should the user indicate that the speed is inappropriate. (By typing a <*break*> character.) The default speed is 300 baud. The optional third argument, *type,* is a character string describing to *getty* what type of terminal is connected to the line in question. *Getty* understands the following types:

none	default
tec	TEC scope
vt61	DEC vt61
vt100	DEC vt100
tektonix	Tektronix
tek	Tektronix
ds40-1	Teletype DS40-1
hp45	Hewlett-Packard HP45
ds40-2b	Teletype DS40-2b

The default terminal is "**none**"; i.e., any crt or normal terminal unknown to the system. The optional fourth argument, *linedisc,* is a character string describing which line discipline to use in communicating with the terminal. *getty* understands the following line disciplines:

full_duplex	default
full	default
transparent	transparent (see *ioctl*(2))
trans	transparent
half_duplex	half duplex
half	half duplex
votrax	votrax

The default is "**full_duplex**"; i.e., line discipline zero.

When given no optional arguments, *getty* sets the speed of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, newline characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause *getty* to attempt the next

speed in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(2)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

In addition to the standard **UNIX** erase and kill characters, '#' and '@', *getty* also understands '\b'. If the user uses a '\b' as a rubout, *getty* sets the standard erase character to backspace and the standard kill character to '@' instead of '#' and '@'.

Getty also understands the "standard" ESS protocols for erasing, killing and aborting a line, and terminating a line. If *getty* sees the ESS erase character, '_', or kill character, '$', or abort character, '&', or the ESS line terminators, '/' or '!', it attempts to set up the terminal into **STDTTY** mode (see *ioctl*(2)), which has those characters as the erase, kill, and line terminator characters. If it doesn't succeed, the standard erase and kill characters will be used.

Finally, *login* is called with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which will place them in the environment. (see *login*(1))

A test option is provided. When *getty* is invoked with the *−t* switch and a file, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs, /etc/issue

SEE ALSO

ct(1), init(1M), login(1), ioctl(2), tty(4), gettydefs(5), inittab(5)

BUGS

While *getty* does understand simple single character quoting conventions, it is not possible to quote the special control characters that *getty* uses to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. Therefore it is not possible to login via *getty* and type a '#', '@', '/', '!', '_', backspace, '^D', or '&' in your response.

NAME

gex − Graphic EXerciser for Tektronix 4014

SYNOPSIS

gex [−]

DESCRIPT ON

Gex (formerly *llx*) is an interactive graphics program which allows a user to initialize and edit graphical data consisting of lines, arcs, circles, text and library symbols, in two line widths and five line styles.

The graphical data buffer is serial and ranges 0-32K data base units.

The optional ' − ' argument causes a time delay, required by some terminals after erase.

The last file reference is remembered and used as the output file when a null output file name is given. If file name ends with **.tk** 4014 scope code is output, otherwise the graphic data buffer is output. Any **.tk** file can be displayed with the *cat* command.

Long displays and listings can be interrupted with the standard UNIX interrupt key.

Commands are single stroke key depressions which send the current cursor position and the command. When additional typed input is required a prompt message is displayed. When the typed input is multi character it must be terminated with **return** and the standard UNIX character erase (#) and line erase (@) can be used to correct typing errors.

The first *DEFINE* (digitize) produces a *butterfly* on the point, the second produces a *dotted box* around the area defined, and the third is taken as a first. With all *DEFINE* area sub-commands *gex* knows only points, not lines; therefore *DELETE*, *MOVE*, *EDIT*, *COPY*, etc., affect only the points on or within the defined area. For *TEXT* the point is the center of the first character. For *CIRCLE* the point is the center point. For *SYMBOL* the point is the reference point of the symbol.

Inactivity for two minutes will cause *gex* to **Time out!**, this is to relax the scope, **return** brings it back to life again.

Gex Commands:

j JUMP to here. Start a new series of vectors.

v VECTOR Draw a line to here.

t TEXT Place a character string with height *SIZE* and angle *ROTATE* here.

c CIRCLE Draw a circle with radius *SIZE* here.

[SELECT SYMBOL Set library symbol mode.

p PLACE SYMBOL Draw the currently selected symbol at scale *SIZE* and angle *ROTATE* here.

e ERASE SCOPE Clean off scope, does not affect data.

g GRID Set Grid snap increment.

a AVAILABLE BUFF Print buffer space usage.

s SIZE Set mode for *TEXT*, *CIRCLE*, *SYMBOL* and *SCALE*.

r ROTATE Set mode for *TEXT*, *ROTAREA* and *SYMBOL*.

l LINE STYLE Set line width and style for *VECTOR*, *ARC*, *CIRCLE* and *EDIT*.

k KILL TEXT DISP Do not display *TEXT*. See 'C' command.

n NEW DISPLAY Erase scope and redraw picture.

q QUERY Print current status (modes). See 'C' command.

x REF GRID Draw tick marks , at current *GRID* points, on edges of scope.

i INPUT FILE Append data from file to graphic buffer.

O OUTPUT TO FILE Write graphic data buffer to file. Output file names should end with '.g'.

o OUTPUT REPORT Print number of graphic bytes sent to scope.

X BIG X Draw scope diagonals.

. LAST DEFINE Restore last (dd) area.

d DEFINE (digitize) Define a point or area. (*Space bar* can be used here)

- **r** RADIUS aRc (dr) Draw an arc or circle.
- **w** WINDOW (dw ddw) Relocate viewing port.
- **x** MINI GRID (dx) Rotated, *ROTATE* degrees, ruler. (ddx) Point matrix.
- **c** COPY (ddc) Make carbon copy, with respect to second (d) point, of graphic data in area.
- **m** MOVE (ddm) Relocate, with respect to second (d) point, graphic data in area. Stacked points (identical X and Y) can be unstacked with two *DEFINE*s on the point.
- **a** ANGLE (dda) Print angle and distance between points. Distance is given in data base units and scope units.
- **f** FLASH (ddf) Redraw the graphic data in area.
- **D** DELETE (ddD) Remove graphic data from buffer.
- **E** EDIT (ddE) Change to current *SIZE*, *ROTATE*, *LINE STYLE* and *SYMBOL* number. See 'C' command.
- **Q** QUERY (ddQ) Print the buffer values for graphic data in area. See *gdump*(1G) for format description.
- *B* ROTAREA (ddR) Rotate, *ROTATE* degrees about second (d) point, all graphic data in area.
- **G** GRID SNAP (ddG) Snap graphic data in area to current **GRID**. Use with care, can cause coincident points!
- **S** SCALE (ddS) Scale graphic data in area by *SIZE* / **100**, leaving low left point fixed.
- **M** MIRROR Reflect area about midline. *TEXT* and some *SYMBOL*s do not mirror correctly.

w WINDOW Alter viewing port.

e,w,n,s EAST, WEST, NORTH, SOUTH

u,d UP, DOWN Make the picture smaller or larger.

- **k** KEEP Save current window for *HOME*.
- **h** HOME Restore saved window.
- **a** ALL Display zero to 32K across scope.
- **r** RATIO TRUNCATE Change display *WINDOW* ratio to integer value.
- **z** LIB SYMB ZONE Zoom to given zone. For editing symbol library files.

? MENU Print this list of commands.

H HOLD AUTO DISP Do not automatically redisplay.

C CHANGE Edit, query or kill modes. See **CHANGE** discussion below.

P PLOT Write a plot file. (not implemented)

! UNIX Temporary escape to UNIX.

b BACKUP Start a clean sheet of paper.

y RESTORE Recovery from inadvertant *BACKUP* or *COPY*. Should be issued immediately after the bad *BACKUP* or *COPY*.

^ DONE Terminate program.

CHANGE modes for *EDIT*, *QUERY* and *KILL*

The *gex* initial default for the '**q**' command prints only lines 1 and 2, (default values are shown below), but can be set with the '**C**' command to print any, all or none of the following lines:

L1: SIZE= 70 ROT= 0 LS= 0,0 SYM= 0 KTX= F HOLD= F
L2: GRID= 10,10 X= 17600 Y= 12500 OF= tx.g
L3: EMODES: TR= T TS= T SR= T SS= T SN= T CR= T LS= T
L4: QMODES: L1= T L2= T L3= F L4= F L5= F L6= F L7= F
L5: KMODES: TX= F SY= F SX= F AX= F AC= F
L6: Low= 11300,11300 High= 21530,18970 WR= 10.000
L7: GXY= 100,100 WINC= 400

Line 1 shows the current *SIZE*, *ROTATE*, *LINE STYLE* (width, style), selected *SYMBOL* number, *KILL TEXT* and *HOLD* flags (*True*,*False*).

Line 2 shows the current *GRID* snap increments (x, y), the cursor position (x, y) mapped to 32K, and the remembered output file name.

Line 3 shows the current *EDIT* modes: Text Rotate, Text Size, Symbol Rotate, Symbol Size, Symbol Number, Circle Radius, and Line Style. The 'KEYS' **TR**, **TS**, ... are used by the '*C*' command to change the *True False* state. Only 'KEYS' in the *True* state are edited, thus setting all seven 'KEYS' to *False* completely disables the *EDIT* command. The 'KEY' **AT** sets all 'KEYS' to *True* and the 'KEY' **AF** sets all 'KEYS' to *False*.

Line 4 shows the current 'query' modes for lines 1 thru 7. The 'KEYS' idea is similar to the *EDIT* mode 'KEYS' above.

Line 5 shows the current 'kill' modes: **TeXt**, **SYmbols**, Symbol reference point **X**, Arc reference point **X** and Arc Center point. The 'KEYS' idea is similar to the *EDIT* mode 'KEYS' above.

Line 6 shows the **Low** and **High** points (x, y) of the current *WINDOW* in data base (0 - 32K) units and the *WINDOW* ratio which is computed = (xhigh -xlow) / 1023.

Line 7 shows the internal *GRID* increments (X, Y) and the *WINDOW* increment used for *WINDOW* moves which is computed =

(internalGX + internalGY) * 2.

and attempts to guess and print the current symbol zone.

GRAPHIC DATA FILE FORMAT

The first word of every *gex* graphic data file contains the value 040154 octal, 16492 decimal. The graphic data is in records consisting of the following fields:

RECORD TYPE	DATA FIELDS					
Vector	X	Y	cntl, **0**			
Jump	X	Y	cntl, **1**			
Text	X	Y	cntl, **2**	rotate	size	text
Circle	X	Y	cntl, **3**	radius		
arc	X	Y	cntl, **4**			
Symbol	X	Y	cntl, **5**	rotate	size	symbol

X and *Y* are 15 bit integers, range 0 thru 32767.

The low byte of *cntl* is shown, the high byte contains the line style which is divided into two sub fields, the low 5 bits are the line style and the high 3 bits are the line width. Valid line styles are **0** solid, **1** dotted, **2** dotdashed, **3** shortdashed, **4** longdashed. Valid line widths are **0** normal, **1** bold.

Rotate, *size*, *radius* and *symbol* are full word integers.

Text is a null terminated byte string.

LIBRARY FILES

When *gex* starts it attempts to open **./lib.g**, if it exists it is used as the source for symbols. If **./lib.g** is not found an attempt is made to open **/usr/lib/gexlib.g**, if this fails *gex* runs without a symbol library. When no library exists or a selected symbol does not exist, a dot is displayed at symbol references. It is recommended that user libraries be kept in a directory unique to each library along with the subordinate graphic files.

Each symbol in a *gex* symbol library is selected via a number in the range 0 thru 4095. This number identifies a particular symbol zone within the 0 thru 32K range of *gex*. The range of each symbol zone is 512 X 512. Since the low left corner of each *WINDOW* is snapped to the current *GRID* the perimeter of the symbol zone, as it appears on the scope, may be part of the adjacent symbol zone. The reference point for each symbol is the first data point of that symbol.

If a picture is worth 1000 words, try this:
(You do what's in quotes)

1. "**chdir**"" to a clean (empty) directory
2. "**gex**" start *gex*
3. "**w**" window, "**z**" zone, "**1000**"
4. draw some stuff (this will be symbol 1000)
5. "**w**" window, "**z** zone, "**2000**
6. draw some more stuff (this will be symbol 2000)
7. "**w**" window, "**a** all
8. notice the tiny versions of the two new symbols!
9. "O" output, "**lib.g**
10. "^" quit
11. "**gex**" restart *gex*
12. "**[**" select symbol, "**1000**
13. "**position cursor, p**" place symbol
14. repeat 13 a couple times
15. "**[**" select symbol, "**2000**
16. "**position cursor, p**" place symbol

17. repeat 16 a couple times
18. got the picture ? if not call for help!

FILES

tx.g	Default output file
/usr/bin/gexmenu	The ? command
lib.g	Symbol library
/usr/lib/gexlib.g	Default library

SEE ALSO

atgex(1G), gcon(1G), gdump(1G), gsplit(1G), hatch(1G), tkdump(1G)

DIAGNOSTICS

Buffer full EXPECT blowup soon
Graphic buffer about to overflow.

Can't create ____
Write permission in directory or file system.

Cannot OPEN _____
Probably name was mistyped or file does not exist.

DATA is on file -- tx.g
Buffer not empty on exit.

DEFINE two points first
This command requires two points.

DELTA ZERO
Cursor was not moved to establish a delta for **COPY** or **MOVE**.

File Format ERROR
Not a *gex* type file.

GRID TOO SMALL
The current **GRID** is too small to display.

INVALID data CH=NNN
Bad data transfer to cpu.

LIMIT
Can't move **WINDOW** any more.

NO CHANGE
Nothing has been added to buffer (y **RESTORE**).

NOTHING FOUND / CHANGED
Nothing was found or what was found did not change.

Not ENOUGH SPACE IN BUFFER
Not enough space to do the file **INPUT** or **COPY**.

POINT OUT OF BOUNDS
SCALE command aborted because a point would go out.

AUTHOR

D. J. Jackowski

BUGS

Buffer size restriction should be eliminated.

GEX Menu

j = JUMP
v = VECTOR
t = TEXT
c = CIRCLE
l = SELECT SYMBOL
p = PLACE SYMBOL
e = ERASE SCOPE
g = GRID SET
a = AVAILABLE BUFF
s = SIZE
r = ROTATE
l = LINE STYLE
k = KILL TEXT DISP
n = NEW DISPLAY
q = QUERY
x = REF GRID
i = INPUT FILE
O = OUTPUT TO FILE
o = OUTPUT REPORT
X = BIG X
. = LAST DEFINE
d = DEFINE (digitize)
dr = RADIUS aRc (dr)
dw = WINDOW (dw ddw)
dx = MINI GRID (dx ddx)
dc = COPY (ddc)
dm = MOVE (ddm)
da = ANGLE (dda)
df = FLASH (ddf)
dD = DELETE (ddD)
dE = EDIT (ddE)
dQ = QUERY (ddQ)
dR = ROTAREA (ddR)
dG = GRID SNAP (ddG)
dS = SCALE (ddS)
dM = MIRROR (ddM)
w = WINDOW
we = EAST
ww = WEST
wn = NORTH
ws = SOUTH
wu = UP
wd = DOWN
wk = KEEP
wh = HOME
wa = ALL
wr = RATIO TRUNCATE
wz = SYMBOL ZONE
? = MENU
H = HOLD AUTO DISP
C = CHANGE E, q or k modes
! = UNIX
b = BACKUP
y = RESTORE
^ = DONE

NAME

gget − get a file from SCCS

SYNOPSIS

gget [options] [files] [directories]

DESCRIPTION

The *gget* command will retrieve the given source files or directories from a file system dedicated to Source Code Control System (SCCS) files. *Gget* is written in Bourne shell-ese and thus runs only on systems supporting the Bourne shell. *Gget* requires two shell variables to be set and either made *global* or *export*ed. The first is *SCCSOURCE*. It is set to be the directory which subtends all SCCS directories of current interest. Thus for the unix source software, *SCCSOURCE* is set to /usr/src/ucb . The second variable is *SUBSYSTEMS*. It is set to the subdirectories of interest in $SCCSOURCE. Thus someone working on the operating system might set the following:

SCCSOURCE=/usr/src/ucb
SUBSYSTEMS="os io sys"

The command line arguments are files, directories or any valid option to the *gget*(1S) command. If a *file* is specified, all $SUBSYSTEMS in $SCCSOURCE are searched in the order of $SUBSYSTEMS for a file named s.*file*. When found, *gget* executes *get*(1S) on the file with the appropriate directories and 's.' prepended to the file name.

If a directory is specified, the user has two options. If −D is not an input argument *gget* retrieves the named file into the current directory. If the −D option is specified *gget* makes the appropriate directory(s), then does a *cd* to the appropriate directory before executing each *get*(1S).

As an example assume the shell variables are set as above.

gget os

causes the following command to be executed:

/usr/bin/get /usr/src/ucb/os

If the −D option is specified then **mkdir os** and **cd os** occur before the /usr/bin/get .

OPTIONS

−D Make directories and change to them as required

FILES

/usr/bin/get

SEE ALSO

get(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2.

NAME

gls − list the directory $SCCSOURCE with input args appended

SYNOPSIS

gls [ls_opts] [dirs]

DESCRIPTION

The *gls* command uses *g_find*(1S) to locate each named input file. It passes all '-' type arguments to the *ls*(1) command with the resulting full pathname from *g_find*. *Gls* is written in Bourne shell-ese and thus runs only on systems supporting the Bourne shell. *Gls* requires two shell variables to be set and either made *global*or*export*ed. The first is *SCCSOURCE*. It is set to be the directory which subtends all SCCS directories of current interest. Thus for the unix source software, *SCCSOURCE* is set to /usr/src/ucb . The second variable is *SUBSYSTEMS*. It is set to the subdirectories of interest in $SCCSOURCE. Thus someone working on the operating system might set the following:

SCCSOURCE=/usr/src/ucb
SUBSYSTEMS="os io sys"

Gls will report back the differences between each named file and its corresponding SCCS file. As an example assume the shell variables are set as above.

gls param.h

causes the following output:

/usr/src/ucb/sys/s.param.h

Gls is essentially the following shell script:

set 'g_find $1'; ls $1

OPTIONS

Any options valid to the *ls*(1) command.

SEE ALSO

g_find(1S)

DIAGNOSTICS

NA E

gmark − mark a subsystem of SCCS files.

SYNOPSIS

gmark [options] [subsystems]

DESCRIPTION

The *gmark* command will mark a subsystem consisting of SCCS files. *Mark* means take a snapshot of all source file SCCS sid numbers of the files which make up the subsystem. *Gmark* is written in Bourne shell-ese and thus runs only on systems supporting the Bourne shell. *Gmark* requires one shell variable to be set and either made *global* or *export*ed, *SCCSOURCE*. It is set to be the directory which heads all SCCS directories of interest. Thus for the unix source software, *SCCSOURCE* is set to '**/usr/src/ucb**'. (On the Columbus system currently Unix source is under **/usr/src/ucb** in the directories **os**, **io**, and **sys**.) The command line arguments are any of the valid *gmark* options and any valid directory path relative to $SCCSOURCE. The options tell *gmark* what to do on this invocation.

As an example assume the shell variable $SCCSOURCE is set as above. Then the following command:

gmark −M .

causes the following to happen: (here 'remembers' means stores in a temporary file.)

1 *gmark* does a *cd* to '**/usr/src/ucb/.**'.

2 *gmark* reads a file called '**s.markfile**' to find all other directories and files which must be marked along with the current directory. If none exists *gmark* goes to step 5;

3 for each directory found, *gmark* does a *cd* to that directory and does a *gmark* on that directory. It then remembers the new SCCS sid of the markfile in the directory;

4 for each file in markfile *gmark* remembers the most recent SCCS sid;

5 for each file in '.' *gmark* remembers the most recent SCCS sid;

6 if '**−HL**' or '**−LL**' type options were specified *gmark* stream edits the temporary file and writes the requested information on the standard output;

7 if the '**−M**' type option was specified *gmark* uses the temporary file to *delta* the **s.markfile** in the directory '.'. If no **s.markfile** exists, *gmark* creates one with the *admin*(1S) command.

The following options are supported:

−M update the **s.markfile** with the current SID's of the named subsystems. The '**−C**' option may also be specified. If '**−C**' is not specified, *gmark* prompts for a comment. This option conflicts with any of the **−H** or **−L** options.

−C'cmt' This option can only be specified when the **−M** option is specified. The comment is any string (imbedded blanks and newlines are allowed).

−L print the **s.markfile** in the named subsystem.

−LL traverse the markfile chain and print each markfile starting from named subsystem. This effectively is the snapshot taken of the subsystem named. Any descriptive information that does not refer to SCCS sid levels comes out on file descriptor 2. The standard output can be *sed*'ed to actually retrieve a previous version of a subsystem. (see *gmget*(1S), *sed*(1S)).

−H print (*prt*(1S)) the history of the markfile in the named subsystem.

−HL traverse the markfile chain and print the history of each markfile in the chain.

−R*N* do any of the above but start at release *N* of the markfile in the named subsystem. In this mode the output of the *gmark* *−L* command is used to determine the SCCS sid of the down chain markfiles to retrieve.

FILES

s.markfile

SEE ALSO

gmget(1S), gget(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2, and are hopefully self explanatory.

BUGS

This command and its associated undocumented command *mark* should be rewritten in C to make it faster.

NAME

gmget − get a file in SCCS

SYNOPSIS

gmget [**−r**SID] subsystem subsystem ...

DESCRIPTION

The *gmget* command will *gget*(1S) a set of files listed in a markfile created by the *gmark*(1S) command. Each file retrieved will have the SID named in the specified release of the markfile. If no release is specified (by the '**−r**' option) then the named release refers to the most recent SID of the markfile for the named subsystem.

SEE ALSO

gget(1S), gmark(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2, and are hopefully self explanatory.

NAME

gprt − prt a file in SCCS

SYNOPSIS

gprt [options] [files]

DESCRIPTION

The *gprt* command will do the *prt*(1S) command on the files named. All input flags are passed directly to the *prt*(1S) command. The files are accessed through the *g_find*(1S) command.

The command line arguments are files or any valid option to the *prt*(1S) command. If a *file* is specified, all $SUBSYSTEMS in $SCCSOURCE are searched in the order of $SUBSYSTEMS for a file named *s.file*. When found, *gdelta* executes *prt*(1S) on the file with the appropriate directories and "s." prepended to the file name.

SEE ALSO

prt(1S), g_find(1S)

DIAGNOSTICS

All diagnostics are printed on file descriptor 2, and are hopefully self explanatory.

NAME

graph − draw a graph

SYNOPSIS

graph [option] ...

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *plot*(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes ", in which case they may be empty or contain blanks and numbers; labels never contain new-lines.

The following options are recognized, each as a separate argument:

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **−x**).

−b Break (disconnect) the graph after each label in the input.

−c Character string given by next argument is default label for each point.

−g Next argument is grid style, **0** no grid, **1** frame with ticks, **2** full grid (default).

−l Next argument is label for graph.

−m Next argument is mode (style) of connecting lines: **0** disconnected, **1** connected (default). Some devices give distinguishable line styles for other small integers (e.g., the Tektronix 4014: **2**=dotted, **3**=dash-dot, **4**=short-dash, **5**=long-dash).

−s Save screen, don't erase before plotting.

−x[l] If l is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

−y[l] Similarly for *y*.

−h Next argument is fraction of space for height.

−w Similarly for width.

−r Next argument is fraction of space to move right before plotting.

−u Similarly to move up before plotting.

−t Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **−s** option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

plot(1G), spline(1G), plot(5)

BUGS

Graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.
The special line types do not work with the versatek, although they do for *plot*(1G).

NAME

grep, egrep, fgrep − search a file for a pattern

SYNOPSIS

grep [option] ... expression [file] ...

egrep [option] ... [expression] [file] ...

fgrep [option] ... [strings] [file]

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular expressions in the style of *ed*(1); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed strings; it is fast and compact. The following options are recognized:

−v All lines but those matching are printed.

−x (Exact) only lines matched in their entirety are printed (*fgrep* only).

−c Only a count of matching lines is printed.

−l Only the names of files with matching lines are listed (once), separated by new-lines.

−n Each line is preceded by its relative line number in the file.

−b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

−e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a −.

−f *file* The regular expression (*egrep*) or string list (*fgrep*) is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters **$**, *, [, ^, |, (,), and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'.

Fgrep searches for lines that contain one of the *strings* separated by new-lines.

Egrep accepts regular expressions as in *ed*(1), except for \(and \), with the addition of:

1. A regular expression followed by + matches one or more occurrences of the regular expression.
2. A regular expression followed by ? matches 0 or 1 occurrences of the regular expression.
3. Two regular expressions separated by | or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses () for grouping.

The order of precedence of operators is [], then * ? +, then concatenation, then | and new-line.

SEE ALSO

ed(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

BUGS

Ideally there should be only one *grep*, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.
Lines are limited to 256 characters; longer lines are truncated.
Egrep does not recognize ranges, such as **[a-z]**, in character classes.

NAME

gsplit – filter to break gex files into pieces

SYNOPSIS

gsplit [**–ctvxyalo**]

DESCRIPTION

Gsplit allows division of *gex*(1G) files into logical sub files.

Options:

–c	selects circles.
–crN	selects circles of radius *N*.
–t	selects text.
–trN	selects text with rotation *N*.
–tsN	selects text of size *N*.
–lwN	selects line width *N*.
–lsN	selects line style *N*.
–v	selects jumps, vectors and arcs.
–xN	selects anything with *x* ordinate *N*.
–yN	selects anything with *y* ordinate *N*.
–a	select all.
–lN	limit output to *N* points. Output will continue past the limit to complete a set of vectors and arcs.
–o	output in reversed sense, normally the selected data types are output, i.e. **–tc** selects text and circles, **–tco** selects all the rest.

In all options, except **–lN**, "*N*" may be a single integer or a pair of comma separated integers to designate a range, i.e. **–ts50,125** selects text sizes 50 thru 125.

SEE ALSO

gex(1G)

DIAGNOSTICS

"Bad input file"	Input file not in GEX format.
"Junk in file"	Unexpected data in input file.

AUTHOR

D. J. Jackowski

BUGS

Probably !

NAME

gtty – get terminal line options

SYNOPSIS

gtty [line ...]

DESCRIPTION

Gtty will get certain I/O options on the lines specified. If no line is given the standard input is used (file descriptor 0). See stty(1) for an explanation of the options reported by gtty.

SEE ALSO

gtty(2), stty(1)

BUGS

On devices which do not allow variable speeds, the input and output speed is meaningless.

NAME

hatch – filter to hatch *gex* files

SYNOPSIS

hatch [**-sN**] [**-pN**] [**-aN**] [**-nN**] [**-cN**] [**-lxiod**] [**-dd**]

DESCRIPTION

Hatch provides an easy way to add hatch lines to closed or semi-closed figures in *gex*(1G) files. Figures consist of a jump and at least two vectors, or a jump and a series of vectors and arcs, and are delimited by a jump, text or circle.

Options:

- **-sN** sets hatch spacing to *N*, default is 100.
- **-pN** sets hatch spacing to be *N* percent of figure size.
- **-aN** sets hatch angle to *N*, default is 45.
- **-nN** sets min number of points to *N*, default is 3.
- **-cN** sets min chord length to *N*, default is 50.
- **-l** long, output all graphic data, normally only hatch lines are output.
- **-x** cross hatch.
- **-i** ignore figure delimiters.
- **-o** omit all open figures.
- **-d** dump (print) hatch point buffer.
- **-dd** dump and put points in output file.

SEE ALSO

gex(1G)

DIAGNOSTICS

"Space too small"	Space option less than 10.
"Bad input file"	Input file not in GEX format.
"Junk in file"	Unexpected data in input file.
"Too many input points"	Too many points or arcs.
"STRIKE -- too many"	Hatch line requires too many break points.

AUTHORS

V. A. Fasciano & D. J. Jackowski

BUGS

Probably !

NAME

help − ask for help

SYNOPSIS

help [arg] ...

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., "ge6", for message 6 from the *get* command).

type 2 Does not contain numerics (as a command, such as *get*)

type 3 Is all numeric (e.g., "212")

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

FILES

The ASCII file searched for the explanatory information for each type of argument is as follows:

type 1 /usr/lib/help/*prefix-of-argument*

type 2 /usr/lib/help/cmds

type 3 /usr/lib/sccs.hf

If the file to be searched for either a type 1 or a type 2 argument does not exist, the search will be attempted on the file for the type 3 argument. In no case, however, will more than one file be searched per argument.

Anyone wishing to modify the files should list out portions of them − the format will be obvious.

DIAGNOSTICS

Use *help* for help.

NAME

hex − translate binary file to ascii hexadecimal

SYNOPSIS

hex file1 file2

DESCRIPTION

Hex will read *file1* outputting into *file2* two characters for every byte read. Each byte is split up into two "nibbles" and each nibble is mapped into one of the sixteen hexadecimal characters (0-F). These characters are then written into *file2* with a new line character ('\n') separating every 64 characters. Upon completion, *hex* will print out an integer checksum which is computed by summing all of the bytes read.

SEE ALSO

unhex(1), ucat(1)

DIAGNOSTICS

argcount
Can't open < *file1* >
Can't open < *file2* >
Read error

NAME

hold – suspend printing of queued line printer jobs

SYNOPSIS

hold type item [item]

DESCRIPTION

Hold suspends the printing of a set of line printer requests. *Type* (either user, printer, or job) indicates the type of *items* which follow. At least one *item* is required.

In the case of *type user,* the *items* are login ids of users with queued jobs. For each specified user, *hold* suspends the printing of all the user's line printer jobs, regardless of the queue in which the request resides. Suspension is immediate, and any currently printing job is interrupted. If the *type* is *printer,* then each *item* is a printer name, and *hold* suspends the currently printing job on each of the named printers, and disallows further printing by that printer until the *printer* is either *start*ed or *release*d. An active printer, when held, maintains control over the job it was printing unless the job is separately *start*ed or *release*d before such action is taken on the printer. For *type job,* each of the specified jobs is inhibited from printing.

SEE ALSO

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

NAME

hyphen – find hyphenated words

SYNOPSIS

hyphen name ...

DESCRIPTION

It finds all of the words in a document which are hyphenated across lines and prints them back at you in a convenient format.

If no *name* is given, the standard input is used. Thus *hyphen* may be used as a filter.

BUGS

Yes, it gets confused, but with no ill effects other than spurious extra output.

NAME

id − print user and group id

SYNOPSIS

id

DESCRIPTION

Id writes a message on the standard output device giving the user and group id of the invoking process. If the effective and real id do not match, both are printed.

SEE ALSO

getuid(2), geteuid(2), getgid(2), getegid(2)

FILES

/etc/group
/etc/passwd

NAME

idump − dump an inode

SYNOPSIS

/etc/idump [−]inumber [...] device [...] [[[−]inumber [...]] device] ...

DESCRIPTION

Idump dumps all information about inode number *inumber* from the specified *device*(s). If the '−' is not specified, *idump* will print the inode structure information.

If the '−' is specified, *idump* throws the actual contents of the file onto the standard output. If the file is a directory, the output is formatted to look like an "**ls -af**".

SEE ALSO

ls(1), dir(5), fs(5)

NAME

infect − Give a virus to another UNIX system

SYNOPSIS

infect [−v] [−clev] [−slev] [−a] [−wdev] [−idays] tn

DESCRIPTION

Infect uses the cu(1C) command to dial the UNIX system with telephone number *tn*, and infects that system with a virus program. The arguments allow you to specify the type of virus. You can give anything from a mild cold, which will only cause occasional sneezes and sniffles, to a virulent plague that will kill the system immediately. If *tn* is −, *infect* gives the virus to its own UNIX system.

Once a UNIX system has been infected with a virus, the only proven way of curing that system is to vaccinate it (see −v). There have been cases of spontaneous remission, but such things are very rare. Attempts to *flog*(1) an infected system will be counter-productive.

Unfortunately, VAX 11/780 based UNIX systems seem to be naturally immune to all viruses that *infect* can produce. It seems that such systems are self-VAXcinated.

−v Causes *infect* to create a vaccine to combat a virus with the indicated characteristics, and to vaccinate the UNIX system. Only certain users may do vaccinations, and those users must pay the author (and how!). Warning: The distinction between virus and vaccine is marginal at best. Vaccinating a UNIX system that doesn't have the virus may do more harm than good (the dreaded Swine Flu Syndrome). Even if the UNIX system does have the virus, the vaccine might still make things worse −− and heaven help you if the vaccine mutates.

−c Level of contageousness: this specifies how likely it will be for the infected system to pass the virus onto another UNIX system. (Vaccines are never contagious.) Recognized levels include:

- **n** Not contagious (default); the virus cannot spread to other systems.
- **l** Low: the virus can only spread to other systems that share memory or disks with the infected system.
- **m** Mild: the virus can spread through DA links, PCL links, or CE's (CE's act as common carriers).
- **h** High: the virus can spread through tapes.
- **v** Virulent: the infected UNIX calls other UNIX systems, via *cu*(1C) or *uucp*(1C), and infects them.
- **x** Carrier: highly contagious, execpt that the infected UNIX system doesn't suffer from the virus itself, but just passes it on to other systems.

−i The incubation period, in days. The system will not suffer from the affects of the virus until the incubation period is over. If 0, the virus takes affect immediately. A high incubation period allows the virus to infect the system's backup tapes. The system is contagious during the incubation period.

−s Severity level: mild, severe (the default), or deadly. A mild virus is like a slight cold: the infected system crashes (or loses files, or exhibits other noxious behavior) once or twice a day. If the system was not in perfect health before, the users of the infected system might not realize that it has been given a virus, execpt through statistical analysis. A severe virus causes crashes (or whatever) at least every hour. While a system is obviously very sick, it is usuable (but just barely). A UNIX system with a deadly virus is totally unusable.

−a If specified, the virus will be acute; otherwise, it will be chronic. A chronic virus stays at

the severity level specified by **−s**, and the symptoms come and go irregularly. An acute virus starts at the severity level selected by **−s**, but slowly gets worse until it becomes deadly. For example, a system can suffer from a mild, chronic virus for years before anyone realizes that it has a virus.

−w This specifies what "device" (hardware or software) the virus will attack. Recognized codes are:

kern	the UNIX systems's kernal (can cause it to pop)
file	the file system (can cause lost an/or scrambled files)
cc	C compiler (causes it to crash and/or generate bad code)
doc	documentation tools (can cause athlete's footnote)
cmds	any and all commands on the UNIX system
su	super-user (root-rot −− the kryptonite disease)
unauth	unauthorized users (usually a benevolent virus!!)
cpu	the obvious
mem	memory (perhaps causing parity errors)
disk	disk drives (can cause disk-head dandruff)
back	backplane and/or unibus (e.g. nagging back-ache)
air	air conditioning system (a rare pneumonic virus)
any	any or all of the above (a wide spectrum virus)

CAVEATS

The viruses are stored in a special, sealed RV01 disk pack, mounted on **/dev/andromeda**. If this pack is broken, the viruses will spread like wildfire (the Pandora Syndrome).

DIAGNOSTICS

Many, but they're always wrong.

BUGS (pardon the pun)

For some unknown reason, attempting to infect a system via a dataswitch connection does not work. Dataswitches have a tendency to intercept viruses first and hog them for themselves.

NOTE:

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

NAME

init – reinitialize line printer demon

SYNOPSIS

init printer name [name]

DESCRIPTION

Init generates a new line printer demon killing any demon which currently exists for the specified printer. At least one printer is required. The purpose of the command is to regenerate the line printer demon, after some printer hardware failure.

SEE ALSO

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

NAME

inews - submit news articles

SYNOPSIS

inews [**-u** [filename]] **-t** title [**-n** [newsgroups]] [**-e** [expiration date]] [**-q**] [**-f** sender]

inews -p [filename]

DESCRIPTION

Inews facilitates the submittal of news articles to the USENET news network. The first form is for submitting user articles. If the **-u** flag is given along with a *filename*, the body will be read from the specified file; Otherwise, the body will be read from the standard input. A *title* must be specified as there is no default. Each article belongs to a list of newsgroups. If the **-n** flag is omitted, the list will default to something like "general". If you wish to submit an article in multiple newsgroups, the *newsgroups* must be seperated by commas and/or spaces. If not specified, the expiration date will be set to the local default. The **-q** flag supresses the question asked when the user creates a new newsgroup. The **-f** flag specifies the article's sender. Without this flag, the sender defaults to the user's name.

The second form is of interest only to netnews installers. It is used internally by netnews when receiving news from another machine on USENET. If *filename* is given, the article will be read from the specified file; Otherwise the article will be read from the standard input. The article may be in either 'A'-prototype format or mail-like format. An expiration date need not be present and a receival date, if present, will be ignored.

After local installation, inews will transmit the article to all systems interested in the article. This is determined by checking the subscription lists of the machines (in /usr/lib/news/sys) against the newsgroups of the article submitted.

FILES

/usr/spool/news/.sys.nnn temporary articles
/usr/spool/news/*newsgroups*/*article_no*. Articles
/usr/lib/news/ngfile List of legal newsgroups
/usr/lib/news/seq Sequence number of last article
/usr/lib/news/history List of all articles ever seen
/usr/lib/news/sys System subscription list

SEE ALSO

mail(1), readnews(1), uucp(1), getdate(3), msgs(1), recnews(1), sendnews(1), uurec(1)

AUTHORS

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

Mark Horton (cbosg!mark)
Bell Laboratories
Columbus, Ohio

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

NAME

init − process control initialization

SYNOPSIS

/etc/init

DESCRIPTION

Init is a general process spawner. Its primary role is to create processes from a script stored in the file **/etc/inittab** (see *inittab*(5)). This file usually has *init* spawn *getty*'s on each line that a user may log in on. It also controls autonomous processes required by any particular system.

Init considers the system to be in a *run level* at any given time. A *run level* can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by *init* for each of these *run levels* is defined in the *inittab* file. *Init* can be in one of eight levels, **0-6** and **S** (**s**). The *run level* is changed by having a privileged user run **/etc/init** (which is linked to **/bin/telinit**). This user spawned *init* sends appropriate signals to the orginal *init* spawned by the operating system when the system was rebooted, telling it which level to change to.

Init is invoked inside UNIX as the last step in the boot procedure. The first thing *init* does is to look for **/etc/inittab** and see if there is an entry of the type *initdefault* (see *inittab*(5)). If there is, *init* uses the level specified in that entry as the initial *run level* to enter. If this entry is not in *inittab* or *inittab* is not found, *init* requests that the user enter a run level from the virtual system console, **/dev/syscon**. If an **S** (**s**). is entered, *init* goes into the *SINGLE USER* level. This is the only level that doesn't require the existence of a properly formated *inittab* file. If **/etc/inittab** doesn't exist, then by default the only legal level that *init* can enter is the *SINGLE USER* level. In the *SINGLE USER* level the virtual console terminal **/dev/syscon** is opened for reading and writing and the command **/bin/su** is invoked immediately. To exit from the *SINGLE USER run level* one of two options can be elected. First, if the shell is terminated (via an end-of-file), *init* will reprompt for a new run level. Second, the *init* or *telinit*(1M) command can signal *init* and force it to change the *run level* of the system.

When attempting to boot the system, failure of *init* to prompt for a new run level may be due to the fact that the device **/dev/syscon** is linked to a device other than the physical system teletype (**/dev/systty**). If this occurs, *init* can be forced to relink **/dev/syscon** by typing a delete on the system teletype which is colocated with the processor.

When *init* prompts for the new run level the operator may only enter one of the digits **0** through **6** or the letters **S** or **s**. If **S** is entered *init* operates as previously described in *SINGLE USER* mode with the additional result that **/dev/syscon** is linked to the user's terminal line, thus making it the virtual system console. A message is generated on the physical console, **/dev/systty**, saying where the virtual terminal has been relocated.

If a **0** through **6** is entered *init* enters the corresponding *run level*. Any other input will be rejected and the user will be reprompted. If this is the first time *init* has entered a *run level* other than *SINGLE USER*, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the level entered matches that of the entry before any normal processing of *inittab* takes place. In this way any special initialization of the operating system,such as mounting file systems, can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that *run level*.

Run level 0 is usually defined by the user to contain all of the terminal processes and daemons that are spawned in the multi-user environment.

In a multi-user environment, the *inittab* file is usually set up so that *init* will create a process for each terminal on the system.

For terminal processes, ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When *init* receives a child death signal,

telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** (and **/etc/wtmp** if it exits) (see *who*(1)). A history of the processes spawned is kept in **/etc/wtmp** if such a file exists.

To spawn each process in the *inittab* file, *init* reads each entry and for each entry which should be respawned, it forks a child process. After it has spawned all of the processes specified by the *inittab* file, *init* waits for one of its descendant processes to die, a powerfail signal, or until *init* is signaled by *init* or *telinit*(1M) to change the system's run level. When one of the above three conditions occurs, *init* re-examines the *inittab* file. New entries can be added to the *inittab* file at any time; however *init* still waits for one of the above three conditions to occur. To provide for an instantaneous response the **"init (telinit) Q"** command can wake *init* to reexamine the *inittab* file.

If *init* receives a *powerfail* signal (*SIGPWR* (*signal*(2))) and is not in *SINGLE USER* mode, it scans *inittab* for special powerfail entries. These entries are invoked (if the levels permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions whenever the operating system experiences a power failure.

When *init* is requested to change run levels (via *telinit*(1M)), *init* sends the warning signal (signal 15) to all processes that are undefined in the target *run level*. *Init* waits 20 seconds before forcibly terminating these processes via the kill signal (signal 9).

FILES

/etc/inittab,
/etc/utmp,
/etc/wtmp,
/dev/syscon,
/dev/systty,
/bin/sh,
/bin/su

SEE ALSO

getty(1M), login(1), sh(1), telinit(1M), inittab(5), utmp(5)

DIAGNOSTICS

If *init* finds that it is continuously respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string, and generate an error message on the system console, and refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab*.

NAME

inode – find inode on disk

SYNOPSIS

inode number [...]

DESCRIPTION

For each number in the argument list, *inode* will calculate and print the location of the inode as a block number with respect to the start of the logical device, plus a byte index from the beginning of the specified block.

The output format is:

[number] found in block *x*, index *y*

for each inode number listed. x and y are both octal numbers

NAME

install − install commands

SYNOPSIS

/etc/install [**−c** dira] [**−f** dirb] [**−i**] [**−n** dirc] [**−o**] [**−s**] file [dirx ...]

DESCRIPTION

Install is a command most commonly used in "makefiles" to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx* ...) are given, *install* will search (using *find*(1)) a set of default directories (**/bin**, **/usr/bin**, **/etc**, **/lib**, and **/usr/lib**, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx* ...) are specified after *file*, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

−c *dira* Installs a new command in the directory specified in *dira*. Looks for *file* in *dira* and installs it there if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the **−s** option.

−f *dirb* Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to **755** and **bin**, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the **−o** or **−s** options.

−i Ignores default directory list, searching only through the given directories (*dirx* ...). May be used alone or with any other options other than **−c** and **−f**.

−n *dirc* If *file* is not found in any of the searched directories, it it put in the directory specified in *dirc*. The mode and owner of the new file will be set to **755** and **bin**, respectively. May be used alone or with any other options other than **−c** and **−f**.

−o If *file* is found, this option saves the 'found' file by copying it to **o***file* in the directory in which it was found. May be used alone or with any other options other than **−c**.

−s Suppresses printing of messages other than error messages. May be used alone or with any other options.

NAME

iostat − report I/O and system statistics

SYNOPSIS

iostat [**−atpisbeocgqmzfdhI**] [Irate] [interval [count]]

DESCRIPTION

Iostat reports statistics kept about various activities within the system.

For each disk, the system counts IO completions and the number of words transferred. Also, each sixtieth of a second, the state of each disk is examined and a tally is made if the disk is active. This tally goes into one of four categories, depending on whether the system is executing in user mode, in 'nice' (background) user mode, in system mode, or idle. From all these numbers and from the known transfer rates of the devices it is possible to determine information such as the degree of IO overlap and average seek times for each device.

The optional *interval* argument causes *iostat* to report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

The interval and count arguments may be repeated in pairs to provide varied lengths and numbers of reporting intervals.

With no flag argument *iostat* reports the elapsed time since boot and gives the percentage of time the system has spent in each of the four categories mentioned above.

There are zillions of options:

−a Turns on all options except for **−p, −s, −z, −m, −f, −d, −h** and **−I**.

−t For typewriters collectively, the input and output character rate is reported.

−p Report the percentage of time spent in system mode, in nice (low priority) user mode, in user mode, and in idle mode.

−i Report, in addition to the percentage of time spent in each of the four categories for the **−p** option, the percentage of time each disk was active (seeking or transferring), the percentage of time any disk was active, the percentage of time spent in 'IO wait' (processor idle, but with a disk active,) and the percentage of time spent in interrupt routines with a priority less than that of the system clock.

−s Report the raw timing information: 24 numbers indicating the percentage of time spent in each of the possible configurations of 4 system states and 8 IO states (3 disks each active or not).

−b For each disk, the number of transfers per minute, the number of transfers per second, the milliseconds per average seek, and the milliseconds per data transfer exclusive of seek time is reported.

−e For this invocation of the program report the elapsed time in minutes since the last report. For the first report, this is the time since boot.

−o Print out the number of times the process table, the text table, the inode table, and the file table have overflowed.

−c Print out the rate and count of interrupts (excluding the system clock,) the average number of milliseconds per interrupt, the rate and count of traps caused by kernel switchable text, calls to the swap routine, the number of forks, the number of executes, the number of disk reads, the number of disk writes, and the number of process switches. The interrupt timing information is valid only if there is no hardware interrupting at or above the priority of the system clock. If there is some other high priority device which interrupts at a constant rate this information can be supplied to

iostat using the **−I** option.

−g Report the percentage of time a buffer was needed and none was on the free list, the percentage of time a block was requested and it was found to already be in core, the rate per second of calls to get block (as well as the total number of calls), the percentage of time a physical IO buffer header was needed and marked busy, and the rate per second of successfull calls to the physio subroutine (as well as the total number of calls).

−q Report the average queue size and percent occupancy for each of three queue: the queue of jobs on the swap device but marked runnable (the swap queue); the queue of jobs in core and marked runnable (the run queue); and the queue of jobs waiting for disk IO other than swap IO to complete (the disk queue). Note that the average reported is an average of only those samples with a non-zero occupancy: to compute the true average queue size the size given must be multiplied by the occupancy.

−m From analyzing the memory map report the number of free 64 byte segments; the average, maximum, and minimum size of the segments; and the total number of segments. Notice that this is the one measurement which is *not* a time average, but an instantaneous report.

−z Suppress the first report in a series so that the cumulative statistics since boot are not given and only interval statistics are given.

−f Provide a form feed between reports.

−d −h The **−d** and **−h** options are used to obtain disk drive cylinder usage and seek distance profiles. Counts are kept for a single drive (defined by variable dk_unit in operating system) of the cylinder desired and the required seek distance in 8 cylinder increments. The **−d** option dumps the data in tabular form. The **−h** option produces a histogram on the printer of the data.

−I Requires the user to supply the optional *Irate* argument which specifies the number of interrupts per second at a priority higher than or equal to the system clock. This rate should not include the system clock.

FILES

/dev/mem, /unix

NAME

join − relational database operator

SYNOPSIS

join [**−n**] [**−t***c*] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is '−', the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on their first fields. There is one line in the output for each pair of lines in *file1* and *file2* that have identical first fields. The output line consists of the common first field, then the rest of the line from *file1*, then the rest of the line from *file2*.

If the optional argument **−n** is used, fields are processed on the numeric value of their first field (as a long). In this case, the files must be sorted in increasing numeric order.

Fields are normally terminated by blank, tab or newline. With option **−t***c*, fields are terminated by character *c* or newline.

When first fields are null, *join* makes a cross-product file.

SEE ALSO

sort(1), comm(1), sed(1)

NAME

kas − assembler for the KMC11 microprocessor

SYNOPSIS

kas [name] [**−o** name1] [**−d** name2]

DESCRIPTION

Kas is an assembler/debugger/loader for the KMC11 microprocessor. The optional argument *name* specifies the input file; default is standard input. The optional argument **−o** indicates that the next argument *name1* will be the output of the assembler; default is **a.out**. The optional argument **−d** indicates that the assembler is to be used in debug mode and that the next argument *name2* is the device file name of the microprocessor. No output file is created in debug mode.

Error diagnostics are written on the standard error output and contain the input file name and line number and a brief description of the error. C Preprocessor control lines to change the file name and line number are recognized. This allows the use of the preprocessor to expand the input before assembly.

FILES

a.out	output object
/dev/kmc?	microprocessor device
/lib/cpp	C Preprocessor

SEE ALSO

kun(1), kmc(4).
Assembler for the DEC KMC11 Microprocessor by L. A. Wehr

NOTES

There is no kmc driver in release 2.0 of CB/UNIX.

NAME

kasb − assembler for the KMC11 microprocessor

SYNOPSIS

kasb [name] [**−o** name1] [**−d** name2]

DESCRIPTION

Kasb is an assembler/debugger/loader for the KMC11 microprocessor. The optional argument *name* specifies the input file; default is standard input. The optional argument **−o** indicates that the next argument *name1* will be the output of the assembler; default is **a.out**. The optional argument **−d** indicates that the assembler is to be used in debug mode and that the next argument *name2* is the device file name of the microprocessor. No output file is created in debug mode.

Error diagnostics are written on the standard error output and contain the input file name and line number and a brief description of the error. C Preprocessor control lines to change the file name and line number are recognized. This allows the use of the preprocessor to expand the input before assembly.

FILES

a.out	output object
/dev/kmc?	microprocessor device
/lib/cpp	C Preprocessor

SEE ALSO

kunb(1), kmc(4).
Assembler for the DEC KMC11 Microprocessor by L. A. Wehr

NAME

kill − send a signal to a process or process group

SYNOPSIS

kill [+ sig] [y] processid ...
kill [- sig] [y] processgrp ...

DESCRIPTION

Kill sends the specified process(es) or group(s) the specified signal. The process number of each asynchronous process started with '&' is reported by the Shell. Process and group numbers can also be found by using *ps*(1).

If no signal is specified, *kill* will send the catchable kill signal (number 15) to the specified processes. If it is desired to send some signal other than 15, the signal number may be given, as the first argument preceded by a + sign. If the first argument is preceded by a - sign, the specified signal is sent to the specified process group(s). In the case of process groups, verification will be requested unless the optional y argument is given.

If *processid* is 0, then all processes belonging to the current user and associated with the same process group are killed. Note that this will result in the signal being sent to the kill command itself as well. To avoid this, send the signal to process group 0 using - rather than a plus sign.

If a negative process number is specified then the signal will be sent to all processes which have the same user id as the sender. If the sender is the super-user, the signal will be send to everyone except process 0 and 1. In both cases verification will be required even in the presence of the y argument.

The signaled process must belong to the current user unless he is the super-user.

SEE ALSO

ps(1), sh(1), kill(2), signal(2)

NAME

kun – un-assembler for the KMC11/DMC11 microprocessor

SYNOPSIS

kun [name] [**–o** name1]

DESCRIPTION

Kun is a un-assembler for the KMC11/DMC11 microprocessors. It produces an output listing, acceptable to the assembler *kas*(1), from the input object.

The optional argument, *name*, specifies the input object, default is standard input. The format of the input is either assembler output (first word magic 0410), or formatted dump (first word magic 0440), or raw dump (anything else). In the first two cases, the header is ignored.

The optional argument **–o** indicates that the next argument, *name1*, is to contain the output listing, default is standard output.

The input object is first scanned to determine branch destinations. Labels will be inserted at these locations with format "L*int*:", where *int* is the octal value of the location in words. Immediate values of instructions are also printed in octal. Page breaks are noted by the labels "P0:", ... , "P3:".

SEE ALSO

kas(1), kmc(4).

NOTES

There is no kmc driver in release 2.0 of CB/UNIX.

NAME

kunb − un-assembler for the KMC11/DMC11 microprocessor

SYNOPSIS

kunb [name] [**−o** name1]

DESCRIPTION

Kunb is a un-assembler for the KMC11/DMC11 microprocessors. It produces an output listing, acceptable to the assembler *kasb*(1), from the input object.

The optional argument, *name*, specifies the input object, default is standard input. The format of the input is either assembler output (first word magic 0410), or formatted dump (first word magic 0440), or raw dump (anything else). In the first two cases, the header is ignored.

The optional argument **−o** indicates that the next argument, *name1*, is to contain the output listing, default is standard output.

The input object is first scanned to determine branch destinations. Labels will be inserted at these locations with format "L*int*:", where *int* is the octal value of the location in words. Immediate values of instructions are also printed in octal. Page breaks are noted by the labels "P0:", ... , "P3:".

SEE ALSO

kasb(1), kmc(4).

NAME

ld − link editor

SYNOPSIS

ld [**−sulxrdnimX**] [**−K** [p]] [**−o** name] [**−T** symbol] [**−V** version] file ...

DESCRIPTION

Ld combines several object programs into one; resolves external references; and searches libraries (as created by *ar*(1)). In the simplest case several object *files* are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the **−r** option must be given to preserve the relocation bits.) The output of *ld* is left on **a.out.** This file is made executable if no errors occurred during the load and the **−r** flag was not specified.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important.

The symbols **_etext, _etext1, _etext2, _etext3, _edata, _end, _com_mmr** and **mmtble** (**etext, etext1, etext2, etext3, edata, end** and **com_mmr** in C and **mmtble** in assembler) are reserved, and should not be used. If any of the reserved symbols are referred to, the value is the first location above the text, the size of the second switchable text area, the size of the third switchable text area, the size of the fourth switchable text area, the first location above initialized data, the first location above all data, the number of common memory management registers, and the location of the mmtble in data space respectively. It is erroneous to define these symbols.

Ld understands several flag arguments which are written preceded by a −. Except for **−l**, they should appear before the file names.

−s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by *strip*(1). This option is turned off if there are any undefined symbols.

−u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

−l This option is an abbreviation for a library name. The option **−l** alone stands for **/lib/libc.a**, which is the standard system library for C and assembly language programs. Option **−l***x* stands for **/lib/lib***x***.a,** where *x* is a string. If that does not exist, *ld* tries **/usr/lib/lib***x***.a** A library is searched when its name is encountered, so the placement of a **−l** is significant.

−x Do not preserve local (non-**.globl**) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

−X Save local symbols except for those whose names begin with 'L'. This option is used by *cc* to discard internally generated labels while retaining symbols local to routines.

−r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics. Successful completion of *ld* with this option is accompanied by an exit status of 1.

−d Force definition of common storage even if the **−r** flag is present.

−n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 4K word boundary following the end of the text.

−i When the output file is executed, the program text and data areas will live in separate address spaces. The only difference between this option and **−n** is that here the data starts at location 0. This option turns off the **−n** option.

−K option is used to build **UNIX** with a text size larger than 65536 bytes. The optional *p* parameter is a number from 1 through 7 indicating how many memory management registers are not changed when switching to an alternate text area. This option along with the order of the object files in lib1.a and lib2.a gives the programmer some control of what routines are in which switchable text area. When building a **UNIX** with switchable text areas, the first object file name in each switchable text area is printed. The default value of p is 7. This switch is good for building the operating system only!

−o The *name* argument after **−o** is used as the name of the *ld* output file, instead of **a.out.**

−m The names of all files and archive members used to create the output file are written to the standard output.

−T The *symbol* argument after **−T** is flagged; up to sixteen symbol names may be specified. Each reference to a flagged symbol is indicated by a printed line containing the type of reference and the referencing module. The symbol name must match exactly, including any initial "_". This option is useful in debugging unreferenced or multiply referenced symbols encountered during the link edit process.

−V The octal number supplied as the *version* argument after **−V** is entered in the header of the output file to indicate the operating system environment expected by the module on execution. **Cc**(1) and **occ**(1) will automatically pass the correct version flag when invoking the loader to specify CB/UNIX Release 2.0 and Release 1.0, respectively. The version specified should be compatible with the libraries used to load the module. See **stamp**(1) for further details.

FILES

/lib/lib?.a	libraries
/usr/lib/lib?.a	more libraries
a.out	output file

SEE ALSO

as(1), ar(1), cc(1), nm(1), occ(1), stamp(1), a.out(5)

DIAGNOSTICS

The message "FINDSYM FAILED TO FIND SYMBOL" usually results from an object file in a switchable text area that has a static routine name or a label that is not .globl if assembler code.

NAME

ldrxboot – load floppy disk second level boot

SYNOPSYS

/etc/ldrxboot [0][1] /etc/rxboot2

DESCRIPTION

This program is used to load the second level floppy bootstrap into floppy blocks 0 (minor blocks 1 - 3) and 500 (minor blocks 0 - 1). The first level bootstrap is located in rxboot.s (/etc/rxboot). The second level bootstrap is uboot.s with the rx flag set and is usually kept in /etc/rxboot2.

FILES

/etc/rxboot
/etc/rxboot2

NAME

lex − generate programs for simple lexical tasks

SYNOPSIS

lex [**−rctvfn**] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file **lex.yy.c** is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in **[abx−z]** to indicate a, b, x, y, and z; and the operators *, +, and ? mean respectively any non-negative number of, any positive number of, and either zero or one occurrences of, the previous character or character class. The character . is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported. The notation r{d,e} in a rule indicates between d and e instances of regular expression r. It has higher precedence than |, but lower than *, ?, +, and concatenation. The character ^ at the beginning of an expression permits a successful match only immediately after a new-line, and the character $ at the end of an expression requires a trailing new-line. The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \. Thus **[a−zA−Z]+** matches a string of letters.

Three subroutines defined as macros are expected: **input()** to read a character; **unput(***c***)** to replace a character read; and **output(***c***)** to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex()**, and the library contains a **main()** which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function **yymore()** accumulates additional characters into the same *yytext*; and the function **yyless(***p***)** pushes back the portion of the string matched beginning at *p*, which should be between *yytext* and *yytext+yyleng*. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes **%%** it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a **%%**, as in YACC. Lines preceding **%%** which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with {}. Note that curly brackets do not imply parentheses; only string substitution is done.

EXAMPLE

```
D       [0−9]
%%
if      printf("IF statement\n");
[a−z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
```

```
                    {
                    case '/': break;
                    case '*': unput('*');
                    default: go to loop;
                    }
            }
```

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

The flags must appear before any files. The flag **−r** indicates Ratfor actions, **−c** indicates C actions and is the default, **−t** causes the **lex.yy.c** program to be written instead to standard output, **−v** provides a one-line summary of statistics of the machine generated, **−f** indicates "faster" compilation, so no packing is done, but it can handle much smaller machines only, **−n** will not print out the − summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

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

%p *n* number of positions is *n* (default 2000)

%n *n* number of states is *n* (500)

%t *n* number of parse tree nodes is *n* (1000)

%a *n* number of transitions is *n* (3000)

The use of one or more of the above automatically implies the **−v** option, unless the **−n** option is used.

SEE ALSO

yacc(1)

M. E. Lesk and E. Schmidt, *LEX − Lexical Analyzer Generator*

BUGS

The Ratfor option is not yet fully operational.

NAME

lfcheck − logical file system (LFS) consistency check and repair

SYNOPSIS

lfcheck [−synH] [lfs_name **]**

DESCRIPTION

Lfcheck audits UNIX logical file systems for consistency and corrects any discrepancies. Since these corrections will, in general, result in a loss of data, the program will request user concurrence for each such action. All questions should be answered by typing **yes** or **no** followed by a <CR>. Entering **yes** will cause the action specified by the question to take place, while a **no** response prevents the action from being taken. Entering **−y** or **−n** instead of yes or no has the same effect as setting the **−y** or **−n** option on the command line. Entering **?** when input is required prints a description of acceptable responses. Entering **!** allows the user to execute shell commands, and entering **quit** allows the user to exit the program (except during critical operations).

The options are entered as a dash (−) followed by the legal option characters in any order. If mutually exclusive options are present, the last specified (rightmost appearance in the character string) is the option taken.

Valid option characters are:

- **s** perform a salvage (repair) of the LFS
- **y** automatically answer all questions **yes**
- **n** automatically answer all questions **no**
- **H** print LFS header (configuration) information.

Note: **y** and **n** are mutually exclusive.

The program consists of six separate phases, some of which are skipped if they are not needed.

Phase one: *lfcheck* prompts for options and the LFS name if not specified on the command line, opens the filesystem as a UNIX file, reads and verifies the LFS header block, and initializes its tables.

Phase two: *lfcheck* examines the range of blocks belonging to each (allocated) logical file, checking for blocks which are outside the logical file system (BAD), belong to more than one logical file (DUP), belong to a file and overlap the overhead area of the logical file system (ALLOC & OVHD), or belong to a file but are not marked as allocated in the free-space bitmap on disk (ALLOC & FREE).

Phase three: is executed only if DUP blocks were found in phase two. This phase locates the first owner (logical file) of each DUP and adds the file to the faulty file list. The identity of the first file to own a block is not saved in phase two and must be determined by an additional pass over the file definition entries. A list of all duplicated blocks and all known owners is then printed.

Phase four: is executed only if the salvage option, **−s**, has *not* been specified (check-only) and compares the disk versions of the freelist and free-space bitmap to locate blocks included in the freelist but not marked free in the bitmap and vice-versa. A summary of the discrepancies is printed along with the number of entries that are used in the freelist.

Phase five: is executed only if the salvage option, **−s** , has been specified and loops through the faulty file list, printing the errors associated with each faulty file and prompting for corrective action. Correction usually consists of

deleting the logical file and freeing its blocks, though certain trivial errors can be corrected by *lfcheck* and the user is so informed and asked whether to *keep* the file instead of whether to delete it.

Phase six: is executed only when a salvage has been specified and builds a new freelist and bitmap based on the internal model of the LFS after corrections. During this phase, blocks that do not belong to any logical file and were not included in the freelist or free-space bitmap (MISSING) are automatically returned to the freelist. The new freelist and bitmap are written out to disk, and a summary of the error, allocated file, and free LF block counts is printed.

NOTE: All information on files and blocks is gathered before any alteration of the LFS is made.

EXAMPLES

In the following sample salvage run, the user's input is in bold print.

lfcheck

```
Options: ?
Valid options are any combination of:
        y − yes to all questions
        n − no to all questions
        s − perform LFS salvage
        c − perform LFS check (default)
        H − print LFS header
or <CR> − same as '−c'
Options: s
LFS Name: ?
A valid LFS name is xxx...xx
where /dev/xxx...xx is the name of the block device containing the LFS
LFS Name: lfs2
          /dev/lfs2
Phase 1: Verify Header and Initialize
Phase 2: Find Structural Errors
Phase 3: Locate First Owners
block =   130 owning lfn's =   2       1
block =   131 owning lfn's =   2       1
Phase 5: Delete Faulty Files
  >>>  Status: A=ALLOC, B=BAD, D=DUP, F=FREE, O=OVHD
lfn =     1     start =   128  size =     4    status = A.DF.
Delete file? ?
yes or no? yes
File deleted and blocks returned.
lfn =     2     start =   130  size =     3    status = ..D..
Delete file? quit
  >>>  Critical point in program -- can't quit now
Delete file? no
File kept.
lfn =     3     start =   400  size =    22    status = A..F.
TRIVIAL ERROR: Keep file? yes
File kept.
lfn =     5     start =  2290  size =    10    status = AB.F.
```

```
Delete file? yes
File deleted and blocks returned.
lfn =     6     start =    100  size =     2     status = A..FO
Delete file? yes
File deleted and blocks returned.
Phase 6: Build New Freelist
  >>>  Freelist contains 3 entries (2.36% full)

#alloc files= 5    (0.04%)      #faulty files= 5    (100.00%)
#dup LF blocks= 2    (0.09%) #dup LF block occurrences= 4
#free LF blocks= 2167 (94.26%)  #files retained= 2    (0.01%)
```

FILES

/dev/*lfs_name*	logical filesystem to be checked
/etc/lmtab	logical file mount table
stdout	DUP report; all reports, errors if redirected
stderr	user prompts and error messages

SEE ALSO

lfmount(1), lfumount(1), mklfs(1)

DIAGNOSTICS

Most error messages are self explanatory. Some errors, such as trying to check a mounted LFS or internal table overflow, require the user to decide whether to continue. In such cases, the decision to continue **may** result in incomplete or inaccurate results.

Lfcheck returns the following exit code values:

0 = normal termination (no LFS errors)
1 = LFS errors
2 = *lfcheck* internal error.

WARNINGS

Mounted logical file systems *cannot* be salvaged.

Checking a mounted logical file system may produce inaccurate reports.

Choosing to continue when *lfcheck* tables overflow during a salvage run prevents the addition of new entries to the overflowed table but does not cause the loss of any current entries. This may leave files on disk that have errors which cannot be detected in subsequent *lfcheck* runs once corrections have been made by the current run. If *lfcheck* were to simply exit on table overflow, however, then a partial repair could not be made and the LFS would have to be reloaded from a backup or rebuilt from scratch. A warning message regarding unrecorded errors is printed on program completion after overflow has occurred.

Once the user has begun to delete files, the option to enter **quit** in reply to a correction prompt is no longer allowed, and the **<rubout>** (interrupt) signal is disabled.

Should the user wish to run a check as a non-interactive process then a blanket reply (**−y** or **−n**) must be specified on the command line along with the LFS name, and output should be redirected.

BUGS

Currently, *lfcheck* does not recover from I/O errors and terminates abnormally when they are encountered.

NAME

lfmount − mount logical file system (LFS)

SYNOPSIS

lfmount [char_dev block_dev [−**r**]]

DESCRIPTION

Lfmount mounts a Logical File System; that is, it associates a UNIX block device with the UNIX character device corresponding to the Logical File System in question. *Lfmount* must be executed before any program attempts to open a LFS. Since it is executed when no LFS is open, access to the LFS software is via a "permanently mounted" pseudo character device called **/dev/lfctl** (minor character device number 255). This same device is used for *lfumount*(1), *lfsync*(1), and *lfupdate*(1). All access to a LFS from user-level programs is by the character device. Thus, making the correspondence between the block device and character device tells the LFS software where the physical storage resides. This character device is specified by a character special file *char_dev* in **/dev**. The block device *block_dev* is also assumed to be in **/dev**, and may refer to physical storage on moving head disks or floppy disks. If the −**r** option is given, the block device is mounted read-only. That is, any LFS command involving file creation, deletion, writing, or switching is illegal.

If no arguments are given, the current mount table is printed. The printout is of the form:

LFS /dev/*char_dev* **maps to /dev/***block_dev*

or:

LFS /DEV/*char_dev* **maps to /dev/***block_dev* ***(Read-only)***

The following example associates block device **lfs3** with character device **lfh0**:

lfmount lfh0 lfs3

The next example associates block device **lfs2** with character device **database1** and makes the mount read-only:

lfmount database1 lfs2 −r

FILES

/etc/lmtab	Mount table
/dev/*char_dev*	Character device, one per LFS
/dev/*block_dev*	Block device, one per LFS
/dev/lfctl	LFS control device (minor device 255)

SEE ALSO

lfumount(1)

DIAGNOSTICS

Diagnostics are given if UNIX returns errors on creating, reading, or writing **/etc/lmtab**, if *char_dev* or *block_dev* are already present in the internal LFS mount table (and thus presumably, in **/etc/lmtab**), if the LFS header stored on *block_dev* cannot be accessed, or if *block_dev* is not a Logical File System. The use of *lfmount* is limited to those with root permission. A diagnostic is given otherwise.

WARNINGS

The mount table **/etc/lmtab** can get out of step with the internal mount table kept by the

operating system. For example, if **/etc/lmtab** is removed, LFS operations may still proceed without any problems, because the LFS software does not read it. However, typing **lfmount** will imply that nothing is mounted. Similar problems exist with the UNIX *mount*(1) command.

No check is made to see if the character device given really refers to the proper device for the LFS software. If an incorrect device is put in the mount table, succeeding LFS commands will simply fail.

The order of the arguments (including the **−r** flag) cannot be varied.

NAME

lfsync − update modified LFS data

SYNOPSIS

lfsync

DESCRIPTION

Lfsync is used to write modified LFS file definition entries to disk. File definition entries are used to store information such as the size and location of a logical file. *Lfsync* is normally used just before halting the machine, i.e, at the same time *sync*(1) is executed. It should only be executed when the system is quiescent; otherwise, data which is being modified by another process will not be flushed to disk. For each mounted LFS, *lfsync* writes the LFS header to disk as well as all modified file definition entries. Typing **lfsync** is identical to typing **lfupdate 0**.

SEE ALSO

lfmount(1), lfupdate(1), lfs(3C)

DIAGNOSTICS

Diagnostics are given for I/O errors in flushing the LFS header and file definition entries to disk.

NAME

lfumount − unmount the logical file system (LFS)

SYNOPSIS

lfumount char_dev

DESCRIPTION

Lfumount unmounts a Logical File System (LFS); i.e., the block device containing the physical storage for the LFS is disassociated from the character device for the LFS in question (see *lfmount*(1)). The character special file *char_dev* is assumed to be in **/dev**. The entry for **/dev**/*char_dev* is also removed from the mount table in **/etc/lmtab**.

FILES

/etc/lmtab	LFS mount table
/dev/*char_dev*	Character special file, one per LFS

SEE ALSO

lfmount(1)

DIAGNOSTICS

Diagnostics are given if UNIX returns errors on opening, reading, or writing **/etc/lmtab**, or if *char_dev* is not present in the internal mount table kept by the LFS software. The use of *lfumount* is limited to the super user. A diagnostic is given otherwise.

WARNINGS

Since **/etc/lmtab** is maintained separately from the internal mount table, they can become inconsistent. Thus, it is possible for *lfumount* to complain that *char_dev* is not in **/etc/lmtab**, although it correctly removed it from the internal table.

NAME

lfupdate − update modified LFS data repetitively

SYNOPSIS

lfupdate time

DESCRIPTION

Lfupdate updates the Logical File System (LFS) header and modified file definition entries for each mounted LFS. This update operation is done every *time* seconds, with *lfupdate* sleeping in the interim. If *time* is 0, the update is done just once, and *lfupdate* will exit immediately. This latter mode of operation is identical to the *lfsync*(1) command.

SEE ALSO

lfsync(1)

DIAGNOSTICS

Diagnostics are given for I/O errors in flushing the LFS header and file definition entries to disk.

NAME

line – get line identification

SYNOPSIS

line

DESCRIPTION

Line gives the name of the user's line in the form **ln***nn* for *nn* in the range 00 to 99. The actual path name is then **/dev/ln***nn*.

DIAGNOSTICS

lnxx if the standard input file is not a typewriter.