IEEE P1003.2 Draft 11.2 – September 1991

Copyright © 1991 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only, and subject to the restrictions contained herein.

Permission is hereby also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copies of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

4.25.7 Extended Description

None.

4.25.8 Exit Status

The `fold` utility shall exit with one of the following values:

0 All input files were processed successfully.

>0 An error occurred.

4.25.9 Consequences of Errors

Default.

4.25.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The `cut` and `fold` utilities can be used to create text files out of files with arbitrary line lengths. The `cut` utility should be used when the number of lines (or records) needs to remain constant. The `fold` utility should be used when the contents of long lines needs to be kept contiguous.

The `fold` utility is frequently used to send text files to line printers that truncate, rather than fold, lines wider than the printer is able to print (usually 80 or 132 column positions.)

Although terminal input in canonical processing mode requires the erase character (frequently set to `<backspace>`) to erase the previous character (not byte or column position), terminal output is not buffered and is extremely difficult, if not impossible, to parse correctly; the interpretation depends entirely on the physical device that will actually display/print/store the output. In all known internationalized implementations, the utilities producing output for mixed column width output assume that a `<backspace>` backs up one column position and outputs enough `<backspace>`s to get back to the start of the character when `<backspace>` is used to provide local line motions to support underlining and emboldening operations. Since `fold` without the `–b` option is dealing with these same constraints, `<backspace>` is always treated as backing up one column position rather than backing up one character.

An example invocation that submits a file of possibly long lines to the line printer (under the assumption that the user knows the line width of the printer to be assigned by `lp`):

```
fold -w 132 bigfile | lp
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

History of Decisions Made

Historical versions of the `fold` utility assumed one byte was one character and occupied one column position when written out. This is no longer always true. Since the most common usage of `fold` is believed to be folding long lines for output to limited-length output devices, this capability was preserved as the default case. The `-b` option was added so that applications could `fold` files with arbitrary length lines into text files that could then be processed by the utilities in this standard. Note that although the width for the `-b` option is in bytes, a line will never be split in the middle of a character. (It is unspecified what happens if a width is specified that is too small to hold a single character found in the input followed by a `<newline>`.)

The use of a hyphen as an option to specify standard input was removed from an earlier draft because it adds no functionality and is not historical practice.

The tab stops are hardcoded to be every eighth column to meet historical practice. No new method of specifying other tab stops was invented.

4.26 `getconf` — Get configuration values

4.26.1 Synopsis

`getconf` *system_var*

`getconf` *path_var pathname*

4.26.2 Description

In the first synopsis form, the `getconf` utility shall write to the standard output the value of the variable specified by the *system_var* operand.

In the second synopsis form, the `getconf` utility shall write to the standard output the value of the variable specified by the *path_var* operand for the path specified by the *pathname* operand.

The value of each configuration variable shall be determined as if it were obtained by calling the function from which it is defined to be available by this standard or by POSIX.1 {8} (see Operands). The value shall reflect conditions in the current operating environment.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.26.3 Options

None.

4.26.4 Operands

The following operands shall be supported by the implementation:

system_var A name of a configuration variable whose value is available from the function defined in 7.8.1 [such as *confstr* () in the C binding], from the POSIX.1 {8} *sysconf* () function, one of the additional POSIX.2 variables described in 7.8.2, to be available from the *sysconf* () function, or a minimum value specified by POSIX.1 {8} or POSIX.2 for one of these variables.

The configuration variables and minimum values listed in the:

— Name column of Table 2-16 (Utility Limit Minimum Values)

— Name column of Table 2-17 (Symbolic Utility Limits)

— Name column of Table 2-18 (Optional Facility Configuration Values)

— Name column of POSIX.1 {8} Table 2-3 (Minimum Values)

— Name column of POSIX.1 {8} Table 2-4 (Run-Time Increasable Values)

— Variable column of POSIX.1 {8} Table 4-2 (Configurable System Variables; except CLK_TCK need not be supported), without the enclosing braces and PATH [corresponding to the *confstr* () name value _CS_PATH] shall be recognized as valid *system_var* operands. The implementation may support additional *system_var* operand values.

path_var A name of a configuration variable whose value is available from the POSIX.1 {8} *pathconf* () function.

The configuration variables listed in the Variable column of the POSIX.1 {8} Table 5-2 (Configurable Pathname Variables), without the enclosing braces, shall be recognized as valid *path_var* operands. The implementation may support additional *path_var* operand values.

pathname A pathname for which the variable specified by *path_var* is to be determined.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6401 ### 4.26.5 External Influences

6402 #### 4.26.5.1 Standard Input

6403 None.

6404 #### 4.26.5.2 Input Files

6405 None.

6406 #### 4.26.5.3 Environment Variables

6407 The following environment variables shall affect the execution of `getconf`:

6408 **LANG** This variable shall determine the locale to use for the
6409 locale categories when both **LC_ALL** and the correspond-
6410 ing environment variable (beginning with **LC_**) do not
6411 specify a locale. See 2.6.

6412 **LC_ALL** This variable shall determine the locale to be used to over-
6413 ride any values for locale categories specified by the set-
6414 tings of **LANG** or any environment variables beginning
6415 with **LC_**.

6416 **LC_CTYPE** This variable shall determine the locale for the interpreta-
6417 tion of sequences of bytes of text data as characters (e.g.,
6418 single- versus multibyte characters in arguments).

6419 **LC_MESSAGES** This variable shall determine the language in which mes-
6420 sages should be written.

6421 #### 4.26.5.4 Asynchronous Events

6422 Default.

6423 ### 4.26.6 External Effects

6424 #### 4.26.6.1 Standard Output

6425 If the specified variable is defined on the system and its value is described to be
6426 available from the function in 7.8.1, its value shall be written in the following for-
6427 mat:

6428 `"%s\n"`, *<value>*

6429 Otherwise, if the specified variable is defined on the system, its value shall be
6430 written in the following format:

6431 `"%d\n"`, *<value>*

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

If the specified variable is valid, but is undefined on the system, `getconf` shall write using the following format:

```
"undefined\n"
```

If the variable name is invalid or an error occurs, nothing shall be written to standard output.

4.26.6.2 Standard Error

Used only for diagnostic messages.

4.26.6.3 Output Files

None.

4.26.7 Extended Description

None.

4.26.8 Exit Status

The `getconf` utility shall exit with one of the following values:

0 The specified variable is valid and information about its current state was written successfully.

>0 An error occurred.

4.26.9 Consequences of Errors

Default.

4.26.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The original need for this utility, and for the *confstr* () function, was to provide a way of finding the configuration-defined default value for the **PATH** environment variable. Since **PATH** can be modified by the user to include directories that could contain utilities replacing the POSIX. 2 standard utilities, shell scripts need a way to determine the system supplied **PATH** environment variable value that contains the correct search path for the standard utilities.

It was later suggested that access to the other variables described here could also be useful to applications.

This example illustrates the value of {NGROUPS _MAX}:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
getconf NGROUPS_MAX
```

This example illustrates the value of {NAME_MAX} for a specific directory:

```
getconf NAME_MAX /usr
```

This example shows how to deal more carefully with results that might be unspecified:

```
if value=$(getconf PATH_MAX /usr); then
        if [ "$value" = "undefined" ]; then
                echo PATH_MAX in /usr is infinite.
        else
                echo PATH_MAX in /usr is $value.
        fi
else
        echo Error in getconf.
fi
```

Note that:

```
sysconf(_SC_POSIX_C_BIND);
```

and:

```
system("getconf POSIX2_C_BIND");
```

in a C program could give different answers. The *sysconf* () call supplies a value that corresponds to the conditions when the program was either compiled or executed, depending on the implementation; the *system* () call to `getconf` always supplies a value corresponding to conditions when the program is executed.

History of Decisions Made

This utility was renamed from `posixconf` during balloting because the new name expresses its purpose more specifically, and does not unduly restrict the scope of application of the utility.

This functionality of this utility would not be adequately subsumed by another command such as

```
grep var /etc/conf
```

because such a strategy would provide correct values for neither those variables that can vary at run-time, nor those that can vary depending on the path.

Previous versions of this utility specified exit status 1 when the specified variable was valid, but not defined on the system. The output string `"undefined"` is now used to specify this case with exit code 0 because so many things depend on an exit code of zero when an invoked utility is successful.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.27 `getopts` — Parse utility options

4.27.1 Synopsis

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

4.27.2 Description

The `getopts` utility can be used to retrieve options and option-arguments from a list of parameters. It shall support the utility argument syntax guidelines 3 through 10, inclusive, described in 2.10.2.

Each time it is invoked, the `getopts` utility shall place the value of the next option in the shell variable specified by the *name* operand and the index of the next argument to be processed in the shell variable **OPTIND** . Whenever the shell is invoked, **OPTIND** shall be initialized to 1.

When the option requires an option-argument, the `getopts` utility shall place it in the shell variable **OPTARG** . If no option was found, or if the option that was found does not have an option-argument, **OPTARG** shall be unset.

If an option character not contained in the *optstring* operand is found where an option character is expected, the shell variable specified by *name* shall be set to the question-mark (?) character. In this case, if the first character in *optstring* is a colon (`:`), the shell variable **OPTARG** shall be set to the option character found, but no output shall be written to standard error; otherwise, the shell variable **OPTARG** shall be unset and a diagnostic message shall be written to standard error. This condition shall be considered to be an error detected in the way arguments were presented to the invoking application, but shall not be an error in `getopts` processing.

If an option-argument is missing:

— If the first character of *optstring* is a colon, the shell variable specified by *name* shall be set to the colon character and the shell variable **OPTARG** shall be set to the option character found.

— Otherwise, the shell variable specified by *name* shall be set to the question-mark character, the shell variable **OPTARG** shall be unset, and a diagnostic message shall be written to standard error. This condition shall be considered to be an error detected in the way arguments were presented to the invoking application, but shall not be an error in `getopts` processing; a diagnostic message shall be written as stated, but the exit status shall be zero.

When the end of options is encountered, the `getopts` utility shall exit with a return value greater than zero; the shell variable **OPTIND** shall be set to the index of the first nonoption-argument, where the first `--` argument is considered to be an option-argument if there are no other nonoption-arguments appearing before it, or the value `$# + 1` if there are no nonoption-arguments; the *name*

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

variable shall be set to the question-mark character. Any of the following shall identify the end of options: the special option `--`, finding an argument that does not begin with a `-`, or encountering an error.

The shell variables **OPTIND** and **OPTARG** shall be local to the caller of `getopts` and shall not be exported by default.

The shell variable specified by the *name* operand, **OPTIND** , and **OPTARG** shall affect the current shell execution environment; see 3.12.

If the application sets **OPTIND** to the value 1, a new set of parameters can be used: either the current positional parameters or new *arg* values. Any other attempt to invoke `getopts` multiple times in a single shell execution environment with parameters (positional parameters or *arg* operands) that are not the same in all invocations, or with an **OPTIND** value modified to be a value other than 1, produces unspecified results.

4.27.3 Options

None.

4.27.4 Operands

The following operands shall be supported by the implementation:

optstring A string containing the option characters recognized by the utility invoking `getopts`. If a character is followed by a colon, the option shall be expected to have an argument, which should be supplied as a separate argument. Applications should specify an option character and its option-argument as separate arguments, but `getopts` shall interpret the characters following an option character requiring arguments as an argument whether or not this is done. An explicit null option-argument need not be recognized if it is not supplied as a separate argument when `getopts` is invoked. [See also the *getopt* () Description in B.7]. The characters question-mark and colon shall not be used as option characters by an application. The use of other option characters that are not alphanumeric produces unspecified results. If the option-argument is not supplied as a separate argument from the option character, the value in **OPTARG** shall be stripped of the option character and the `'-'`. The first character in *optstring* shall determine how `getopts` shall behave if an option character is not known or an option-argument is missing. See 4.27.2.

name The name of a shell variable that shall be set by the `getopts` utility to the option character that was found. See 4.27.2.

The `getopts` utility by default shall parse positional parameters passed to the invoking shell procedure. If *arg* s are given, they shall be parsed instead of the positional parameters.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.27.5 External Influences

4.27.5.1 Standard Input

None.

4.27.5.2 Input Files

None.

4.27.5.3 Environment Variables

The following environment variables shall affect the execution of `getopts`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

OPTIND This variable shall be used by the `getopts` utility as the index of the next argument to be processed.

4.27.5.4 Asynchronous Events

Default.

4.27.6 External Effects

4.27.6.1 Standard Output

None.

4.27.6.2 Standard Error

Whenever an error is detected and the first character in the *optstring* operand is not a colon (`:`), a diagnostic message shall be written to standard error with the following information in an unspecified format:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— The invoking program name shall be identified in the message. The invoking program name shall be the value of the shell special parameter `0` (see 3.5.2) at the time the `getopts` utility is invoked. A name equivalent to

```
basename "$0"
```

may be used.

— If an option is found that was not specified in *optstring,* this error shall be identified and the invalid option character shall be identified in the message.

— If an option requiring an option-argument is found, but an option-argument is not found, this error shall be identified and the invalid option character shall be identified in the message.

4.27.6.3 Output Files

None.

4.27.7 Extended Description

None.

4.27.8 Exit Status

The `getopts` utility shall exit with one of the following values:

0 An option, specified or unspecified by *optstring* , was found.

>0 The end of options was encountered or an error occurred.

4.27.9 Consequences of Errors

Default.

4.27.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The `getopts` utility was chosen in preference to the `getopt` utility specified in System V because `getopts` handles option-arguments containing `<blank>` characters.

Since `getopts` affects the current shell execution environment, it is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
(getopts abc value "$@")
nohup getopts ...
find . -exec getopts ... \;
```

it will not affect the shell variables in the caller's environment.

Note that shell functions share **OPTIND** with the calling shell even though the positional parameters are changed. Functions that want to use `getopts` to parse their arguments will usually want to save the value of **OPTIND** on entry and restore it before returning. However, there will be cases when a function will want to change **OPTIND** for the calling shell.

The following example script parses and displays its arguments:

```
aflag=
bflag=
while getopts ab: name
do
        case $name in
        a)      aflag=1;;
        b)      bflag=1
                bval="$OPTARG";;
        ?)      printf "Usage: %s: [-a] [-b value] args\n" $0
                exit 2;;
        esac
done
if [ ! -z "$aflag" ]; then
        printf "Option -a specified\n"
fi
if [ ! -z "$bflag" ]; then
        printf 'Option -b "%s" specified\n' "$bval"
fi
shift $(($OPTIND - 1))
printf "Remaining arguments are: %s\n" "$*"
```

History of Decisions Made

The **OPTARG** variable is not mentioned in the Environment Variables subclause because it does not affect the execution of `getopts`; it is one of the few ''output-only'' variables used by the standard utilities.

Use of colon (`:`) as an option character (in a previous draft) was new behavior and violated the syntax guidelines. Many objectors felt that it did not add enough to `getopts` to warrant mandating the extension to existing practice. The colon is now specified to behave as in the KornShell version of the `getopts` utility; when used as the first character in the *optstring* operand, it disables diagnostics concerning missing option-arguments and unexpected option characters. This replaces the use of the **OPTERR** variable that was specified in an earlier draft.

The formats of the diagnostic messages produced by the `getopts` utility and the *getopt* () function are not fully specified because implementations with superior (''friendlier'') formats objected to the formats used by some historical implementations. It was felt to be important that the information in the messages used be

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

uniform between `getopts` and *getopt* (). Exact duplication of the messages might not be possible, particularly if a utility is built on another system that has a different *getopt* () function, but the messages must have specific information included so that the program name, invalid option character, and type of error can be distinguished by a user.

Only a rare application program will intercept a `getopts` standard error message and want to parse it. Therefore, implementations are free to choose the most usable messages they can devise. The following formats are used by many historical implementations:

> `"%s: illegal option -- %c\n"`, <*program name*> , <*option character*>
>
> `"%s: option requires an argument -- %c\n"`, <*program name*> , <*option character*>

Historical shells with built-in versions of *getopt* () or `getopts` have used different formats, frequently not even indicating the option character found in error.

4.28 `grep` — File pattern searcher

4.28.1 Synopsis

`grep [ –E | –F ][ –c | –l | –q ][–insvx] –e` *pattern_list* `... [–f` *pattern_file* `]` ... `[`*file* ... `]`

`grep [ –E | –F ][ –c | –l | –q ][–insvx][–e` *pattern_list* `]... –f` *pattern_file* ... `[`*file* ... `]`

`grep [ –E | –F ][ –c | –l | –q ][–insvx]` *pattern_list* `[`*file* ... `]`

Obsolescent Versions :

`egrep [ –c | –l ][–inv] –e` *pattern_list* `[`*file* ... `]`

`egrep [ –c | –l ][–inv] –f` *pattern_file* `[`*file* ... `]`

`egrep [ –c | –l ][–inv]` *pattern_list* `[`*file* ... `]`

`fgrep [ –c | –l ][–invx] –e` *pattern_list* `[`*file* ... `]`

`fgrep [ –c | –l ][–invx] –f` *pattern_file* `[`*file* ... `]`

`fgrep [ –c | –l ][–invx]` *pattern_list* `[`*file* ... `]`

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.28.2 Description

The `grep` utility shall search the input files, selecting lines matching one or more patterns; the types of patterns shall be controlled by the options specified. The patterns are specified by the `-e` option, `-f` option, or the *pattern_list* operand. The *pattern_list* 's value shall consist of one or more patterns separated by `<newline>`s; the *pattern_file* 's contents shall consist of one or more patterns terminated by `<newline>`s. By default, an input line shall be selected if any pattern, treated as an entire basic regular expression (BRE) as described in 2.8.3, matches any part of the line; a null BRE shall match every line. By default, each selected input line shall be written to the standard output.

Regular expression matching shall be based on text lines. Since `<newline>` separates or terminates patterns (see the `-e` and `-f` options below), regular expressions cannot contain a `<newline>` character. Similarly, since patterns are matched against individual lines of the input, there is no way for a pattern to match a `<newline>` found in the input.

A command invoking the (obsolescent) `egrep` utility with the `-e` option specified shall be equivalent to the command:

```
grep -E [ -c | -l ] [-inv] -e pattern_list [file ... ]
```

A command invoking the `egrep` utility with the `-f` option specified shall be equivalent to the command:

```
grep -E [ -c | -l ] [-inv] -f pattern_file [file ... ]
```

A command invoking the `egrep` utility with the *pattern_list* specified shall be equivalent to the command:

```
grep -E [ -c | -l ] [-inv] pattern_list [file ... ]
```

A command invoking the (obsolescent) `fgrep` utility with the `-e` option specified shall be equivalent to the command:

```
grep -F [ -c | -l ] [-invx] -e pattern_list [file ... ]
```

A command invoking the `fgrep` utility with the `-f` option specified shall be equivalent to the command:

```
grep -F [ -c | -l ] [-invx] -f pattern_file [file ... ]
```

A command invoking the `fgrep` utility with the *pattern_list* operand specified shall be equivalent to the command:

```
grep -F [ -c | -l ] [-invx] pattern_list [file ... ]
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.28.3 Options

The `grep` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`-E` Match using extended regular expressions. Treat each pattern specified as an ERE, as described in 2.8.4. If any entire ERE pattern matches an input line, the line shall be matched. A null ERE shall match every line.

`-F` Match using fixed strings. Treat each pattern specified as a string instead of a regular expression. If an input line contains any of the patterns as a contiguous sequence of bytes, the line shall be matched. A null string shall match every line.

`-c` Write only a count of selected lines to standard output.

`-e` *pattern_list*
Specify one or more patterns to be used during the search for input. Patterns in *pattern_list* shall be separated by a `<newline>`. A null pattern can be specified by two adjacent `<newline>`s in *pattern_list* ; in the obsolescent forms, adjacent `<newline>`s in *pattern_list* produce undefined results. Unless the `-E` or `-F` option is also specified, each pattern shall be treated as a BRE, as described in 2.8.3. In the nonobsolescent forms, multiple `-e` and `-f` options shall be accepted by the `grep` utility. All of the specified patterns shall be used when matching lines, but the order of evaluation is unspecified.

`-f` *pattern_file*
Read one or more patterns from the file named by the pathname *pattern_file*. Patterns in *pattern_file* shall be terminated by a `<newline>`. A null pattern can be specified by an empty line in *pattern_file*. Unless the `-E` or `-F` option is also specified, each pattern shall be treated as a BRE, as described in 2.8.3.

`-i` Perform pattern matching in searches without regard to case. See 2.8.2.

`-l` (The letter ell.) Write only the names of files containing selected lines to standard output. Pathnames shall be written once per file searched. If the standard input is searched, a pathname of `"(standard input)"` shall be written, in the POSIX Locale. In other locales, `standard input` may be replaced by something more appropriate in those locales.

`-n` Precede each output line by its relative line number in the file, each file starting at line 1. The line number counter shall be reset for each file processed.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6783 `-q` Quiet. Do not write anything to the standard output, regardless
6784 of matching lines. Exit with zero status if an input line is
6785 selected.

6786 `-s` Suppress the error messages ordinarily written for nonexistent or
6787 unreadable files. Other error messages shall not be suppressed.

6788 `-v` Select lines not matching any of the specified patterns. If the `-v`
6789 option is not specified, selected lines shall be those that match
6790 any of the specified patterns.

6791 `-x` Consider only input lines that use all characters in the line to
6792 match an entire fixed string or regular expression to be matching
6793 lines.

6794 ### 4.28.4 Operands

6795 The following operands shall be supported by the implementation:

6796 *pattern* Specify one or more patterns to be used during the search for
6797 input. This operand shall be treated as if it were specified as
6798 `-e` *pattern_list* (see 4.28.3).

6799 *file* A pathname of a file to be searched for the pattern(s). If no *file*
6800 operands are specified, the standard input shall be used.

6801 ### 4.28.5 External Influences

6802 #### 4.28.5.1 Standard Input

6803 The standard input shall be used only if no *file* operands are specified. See Input
6804 Files.

6805 #### 4.28.5.2 Input Files

6806 The input files shall be text files.

6807 #### 4.28.5.3 Environment Variables

6808 The following environment variables shall affect the execution of `grep`:

6809 **LANG** This variable shall determine the locale to use for the
6810 locale categories when both **LC_ALL** and the correspond-
6811 ing environment variable (beginning with **LC_**) do not
6812 specify a locale. See 2.6.

6813 **LC_ALL** This variable shall determine the locale to be used to over-
6814 ride any values for locale categories specified by the set-
6815 tings of **LANG** or any environment variables beginning
6816 with **LC_**.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6817 **LC_COLLATE** This variable shall determine the locale for the behavior of
6818 ranges, equivalence classes, and multicharacter collating
6819 elements within regular expressions.

6820 **LC_CTYPE** This variable shall determine the locale for the interpreta-
6821 tion of sequences of bytes of text data as characters (e.g.,
6822 single- versus multibyte characters in arguments) and the
6823 behavior of character classes within regular expressions.

6824 **LC_MESSAGES** This variable shall determine the language in which mes-
6825 sages should be written.

6826 #### 4.28.5.4 Asynchronous Events

6827 Default.

6828 ### 4.28.6 External Effects

6829 #### 4.28.6.1 Standard Output

6830 If the `-l` option is in effect, and the `-q` option is not, a single output line shall be
6831 written for each file containing at least one selected input line:

6832 `"%s\n"`, *file*

6833 Otherwise, if more than one *file* argument appears, and `-q` is not specified, the
6834 `grep` utility shall prefix each output line by:

6835 `"%s:"`, *file*

6836 The remainder of each output line shall depend on the other options specified:

6837 — If the `-c` option is in effect, the remainder of each output line shall contain:

6838 `"%d\n"`, <*count*>

6839 — Otherwise, if `-c` is not in effect and the `-n` option is in effect, the following
6840 shall be written to standard output:

6841 `"%d:"`, <*line number*>

6842 — Finally, the following shall be written to standard output:

6843 `"%s"`, <*selected-line contents*>

6844 #### 4.28.6.2 Standard Error

6845 Used only for diagnostic messages.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6846 #### 4.28.6.3 Output Files

6847 None.

6848 ### 4.28.7 Extended Description

6849 None.

6850 ### 4.28.8 Exit Status

6851 The `grep` utility shall exit with one of the following values:

6852 0 One or more lines were selected.

6853 1 No lines were selected.

6854 >1 An error occurred.

6855 ### 4.28.9 Consequences of Errors

6856 If the `-q` option is specified, the exit status shall be zero if an input line is
6857 selected, even if an error was detected. Otherwise, default actions shall be per-
6858 formed.

6859 ### 4.28.10 Rationale. *(This subclause is not a part of P1003.2)*

6860 **Examples, Usage**

6861 This `grep` has been enhanced in an upward-compatible way to provide the exact
6862 functionality of the historical `egrep` and `fgrep` commands as well. It was the
6863 clear intention of the working group to consolidate the three `grep`s into a single
6864 command.

6865 The old `egrep` and `fgrep` commands are likely to be supported for many years to 1
6866 come as implementation extensions, allowing existing applications to operate
6867 unmodified.

6868 To find all uses of the word `Posix` (in any case) in the file `text.mm`, and write
6869 with line numbers:

6870
```
grep -i -n posix text.mm
```

6871 To find all empty lines in the standard input: 2

6872
```
grep ^$
```
6873 or
6874
```
grep -v .
```

6875 Both of the following commands print all lines containing strings `abc` or `def` or
6876 both:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
grep -E 'abc
def'

grep -F 'abc
def'
```

Both of the following commands print all lines matching exactly `abc` or `def`:

```
grep -E '^abc$
^def$'

grep -F -x 'abc
def'
```

History of Decisions Made

The *−e pattern_list* option has the same effect as the *pattern_list* operand, but is useful when *pattern_list* begins with the hyphen delimiter. It is also useful when it is more convenient to provide multiple patterns as separate arguments.

Earlier drafts did not show that the `−c`, `−l`, and `−q` options were mutually exclusive. This has been fixed to more closely align with historical practice and documentation.

Historical implementations usually silently ignored all but one of multiply specified `−e` and `−f` options, but were not consistent as to which specification was actually used.

POSIX.2 requires that the nonobsolescent forms accept multiple `−e` and `−f` options and use all of the patterns specified while matching input text lines. [Note that the order of evaluation is not specified. If an implementation finds a null string as a pattern, it is allowed to use that pattern first (matching every line) and effectively ignore any other patterns.]

The `−b` option was removed from the Options subclause, since block numbers are implementation dependent.

The System V restriction on using `−` to mean standard input was lifted.

A definition of action taken when given a null RE or ERE is specified. This is an error condition in some historical implementations.

The `−l` option previously indicated that its use was undefined when no files were explicitly named. This behavior was historical and placed an unnecessary restriction on future implementations. It has been removed.

The `−q` option was added at the suggestion of members of the balloting group as a means of easily determining whether or not a pattern (or string) exists in a group of files. When searching several files, it provides a performance improvement (because it can quit as soon as it finds the first match) and requires less care by the user in choosing the set of files to supply as arguments (because it will exit zero if it finds a match even if `grep` detected an access or read error on earlier file operands).

The historical BSD `grep −s` option practice is easily duplicated by redirecting standard output to `/dev/null`. The `−s` option required here is from System V.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The −x option, historically available only with `fgrep`, is available here for all of the nonobsolescent versions.

4.29 `head` — Copy the first part of files

4.29.1 Synopsis

`head [−n` *number* `] [`*file* ... `]`

Obsolescent version:

`head [−`*number* `] [`*file* ... `]`

4.29.2 Description

The `head` utility shall copy its input files to the standard output, ending the output for each file at a designated point.

Copying shall end at the point in each input file indicated by the `−n` *number* option (or the obsolescent version's *−number* argument). The option-argument *number* shall be counted in units of lines.

4.29.3 Options

The `head` utility shall conform to the utility argument syntax guidelines described in standard described in 2.10.2, except that the obsolescent version accepts multicharacter numeric options.

The following option shall be supported by the implementation in the nonobsolescent version:

`−n` *number* The first *number* lines of each input file shall be copied to standard output. The *number* option argument shall be a positive decimal integer.

If no options are specified, `head` shall act as if `-n 10` had been specified.

In the obsolescent version, the following option shall be supported by the implementation:

−number The *number* argument is a positive decimal integer with the same effect as the `−n`*number* option in the nonobsolescent version.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.29.4 Operands

The following operand shall be supported by the implementation:

file A pathname of an input file. If no *file* operands are specified, the standard input shall be used.

4.29.5 External Influences

4.29.5.1 Standard Input

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

4.29.5.2 Input Files

Input files shall be text files, but the line length shall not be restricted to {LINE _MAX} bytes.

4.29.5.3 Environment Variables

The following environment variables shall affect the execution of `head`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

4.29.5.4 Asynchronous Events

Default.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.