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All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change.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]

11023 ### 4.53.4 Operands

11024 The following operand shall be supported by the implementation:

11025 *file* A pathname of a directory entry to be removed.

11026 ### 4.53.5 External Influences

11027 #### 4.53.5.1 Standard Input

11028 Used to read an input line in response to each prompt specified in 4.53.6.1. Oth-
11029 erwise, the standard input shall not be used.

11030 #### 4.53.5.2 Input Files

11031 None.

11032 #### 4.53.5.3 Environment Variables

11033 The following environment variables shall affect the execution of `rm`:

11034 **LANG** This variable shall determine the locale to use for the
11035 locale categories when both **LC_ALL** and the correspond-
11036 ing environment variable (beginning with **LC_**) do not
11037 specify a locale. See 2.6.

11038 **LC_ALL** This variable shall determine the locale to be used to over-
11039 ride any values for locale categories specified by the set-
11040 tings of **LANG** or any environment variables beginning
11041 with **LC_**.

11042 **LC_COLLATE** This variable shall determine the locale for the behavior of
11043 ranges, equivalence classes, and multicharacter collating
11044 elements used in the extended regular expression defined
11045 for the `yesexpr` locale keyword in the LC_MESSAGES
11046 category.

11047 **LC_CTYPE** This variable shall determine the locale for the interpreta-
11048 tion of sequences of bytes of text data as characters (e.g.,
11049 single- versus multibyte characters in arguments) and the
11050 behavior of character classes within regular expressions
11051 used in the extended regular expression defined for the
11052 `yesexpr` locale keyword in the LC_MESSAGES category.

11053 **LC_MESSAGES** This variable shall determine the processing of affirmative
11054 responses and the language in which messages should be
11055 written.

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

11056 #### 4.53.5.4 Asynchronous Events

11057 Default.

11058 ### 4.53.6 External Effects

11059 #### 4.53.6.1 Standard Output

11060 None.

11061 #### 4.53.6.2 Standard Error

11062 Prompts shall be written to standard error under the conditions specified in
11063 4.53.2 and 4.53.3. The prompts shall contain the *file* pathname, but their format
11064 is otherwise unspecified. The standard error shall also be used for diagnostic
11065 messages.

11066 #### 4.53.6.3 Output Files

11067 None.

11068 ### 4.53.7 Extended Description

11069 None.

11070 ### 4.53.8 Exit Status

11071 The `rm` utility shall exit with one of the following values:

11072 0 If the `-f` option was not specified, all the named directory entries were
11073 removed; otherwise, all the existing named directory entries were
11074 removed.

11075 >0 An error occurred.

11076 ### 4.53.9 Consequences of Errors

11077 Default.

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

11078 **4.53.10 Rationale.** *(This subclause is not a part of P1003.2)*

11079 **Examples, Usage**

11080 The *SVID* requires that systems do not permit the removal of the last link to an
11081 executable binary file that is being executed. Thus, the `rm` utility can fail to
11082 remove such files.

11083 The `-i` option causes `rm` to prompt and read the standard input even if the stan-
11084 dard input is not a terminal, but in the absence of `-i` the mode prompting is not
11085 done when the standard input is not a terminal.

11086 For absolute clarity, paragraphs (2)(b) and (3) in 4.53.2, describing `rm`'s behavior
11087 when prompting for confirmation, should be interpreted in the following manner:

```
11088     if ((NOT f_option) AND
11089             ((not_writable AND input_is_terminal) OR i_option))
```

11090 It is forbidden to remove the names dot and dot-dot in order to avoid the conse-
11091 quences of inadvertently doing something like:

```
11092     rm -r .*
```

11093 The following command

```
11094     rm a.out core
```

11095 removes the directory entries `a.out` and `core`.

11096 The following command

```
11097     rm -Rf junk
```

11098 removes the directory `junk` and all its contents, without prompting.

11099 **History of Decisions Made**

11100 The exact format of the interactive prompts is unspecified. Only the general
11101 nature of the contents of prompts are specified, because implementations may
11102 desire more descriptive prompts than those used on historical implementations.
11103 Therefore, an application not using the `-f` option, or using the `-i` option relies on
11104 the system to provide the most suitable dialogue directly with the user, based on
11105 the behavior specified.

11106 The `-r` option is existing practice on all known systems. The synonym `-R` option
11107 is provided for consistency with the other utilities in this standard that provide
11108 options requesting recursive descent.

11109 The behavior of the `-f` option in historical versions of `rm` is inconsistent. In gen-
11110 eral, along with ''forcing'' the unlink without prompting for permission, it always
11111 causes diagnostic messages to be suppressed and the exit status to be unmodified
11112 for nonexistent operands and files that cannot be unlinked. In some versions,
11113 however, the `-f` option suppresses usage messages and system errors as well.
11114 Suppressing such messages is not a service to either shell scripts or users.

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

It is less clear that error messages regarding unlinkable files should be suppressed. Although this is historical practice, this standard does not permit the `-f` option to suppress such messages.

When given the `-r` and `-i` options, historical versions of `rm` prompt the user twice for each directory, once before removing its contents and once before actually attempting to delete the directory entry that names it. This allows the user to ''prune'' the file hierarchy walk. Historical versions of `rm` were inconsistent in that some did not do the former prompt for directories named on the command line and others had obscure prompting behavior when the `-i` option was specified and the permissions of the file did not permit writing. The POSIX.2 `rm` differs little from historic practice, but does require that prompts be consistent. Historical versions of `rm` were also inconsistent in that prompts were done to both standard output and standard error. POSIX.2 requires that prompts be done to standard error, for consistency with `cp` and `mv` and to allow existing extensions to `rm` that provide an option to list deleted files on standard output.

The `rm` utility is required to descend to arbitrary depths so that any file hierarchy may be deleted. This means, for example, that the `rm` utility cannot run out of file descriptors during its descent, i.e., if the number of file descriptors is limited, `rm` cannot be implemented in the historical fashion where a file descriptor is used per directory level. Also, `rm` is not permitted to fail because of path length restrictions, unless an operand specified by the user is longer than {PATH_MAX}.

4.54 `rmdir` — Remove directories

4.54.1 Synopsis

`rmdir [-p]` *dir* ...

4.54.2 Description

The `rmdir` utility shall remove the directory entry specified by each *dir* operand, which shall refer to an empty directory.

Directories shall be processed in the order specified. If a directory and a subdirectory of that directory are specified in a single invocation of the `rmdir` utility, the subdirectory shall be specified before the parent directory so that the parent directory will be empty when the `rmdir` utility tries to remove it.

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

4.54.3 Options

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

The following option shall be supported by the implementation:

`-p` Remove all directories in a pathname. For each *dir* operand:

(1) The directory entry it names shall be removed.

(2) If the *dir* operand includes more than one pathname component, effects equivalent to the following command shall occur:

```
rmdir -p $(dirname dir )
```

4.54.4 Operands

The following operand shall be supported by the implementation:

dir A pathname of an empty directory to be removed.

4.54.5 External Influences

4.54.5.1 Standard Input

None.

4.54.5.2 Input Files

None.

4.54.5.3 Environment Variables

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

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).

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

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

4.54.5.4 Asynchronous Events

Default.

4.54.6 External Effects

4.54.6.1 Standard Output

None.

4.54.6.2 Standard Error

Used only for diagnostic messages.

4.54.6.3 Output Files

None.

4.54.7 Extended Description

None.

4.54.8 Exit Status

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

0 Each directory entry specified by a *dir* operand was removed successfully.

>0 An error occurred.

4.54.9 Consequences of Errors

Default.

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

11197 **4.54.10 Rationale.** *(This subclause is not a part of P1003.2)*

11198 **Examples, Usage**

11199 On historical System V systems, the −p option also caused a message to be writ-
11200 ten to the standard output. The message indicated whether the whole path was
11201 removed or part of the path remains for some reason. The Standard Error sub-
11202 clause requires this diagnostic when the entire path specified by a *dir* operand is
11203 not removed, but does not allow the status message reporting success to be writ-
11204 ten as a diagnostic.

11205 If a directory `a` in the current directory is empty except it contains a directory `b`
11206 and `a/b` is empty except it contains a directory `c`,

11207 ```
rmdir -p a/b/c
```

11208 will remove all three directories.

11209 The `rmdir` utility on System V also included an −s option that suppressed the
11210 informational message output by the −p option. This option has been omitted
11211 because the informational message is not specified by POSIX. 2.

## 11212 4.55 `sed` — Stream editor

### 11213 4.55.1 Synopsis

11214 `sed [−n]` *script* `[`*file* ... `]`

11215 `sed [−n][−e` *script* `] ... [−f` *script_file* `] ... [`*file* ... `]`

### 11216 4.55.2 Description

11217 The `sed` utility is a stream editor that shall read one or more text files, make edit-
11218 ing changes according to a script of editing commands, and write the results to
11219 standard output. The script shall be obtained from either the *script* operand
11220 string or a combination of the option-arguments from the −e *script* and
11221 −f *script_file* options.

### 11222 4.55.3 Options

11223 The `sed` utility shall conform to the utility argument syntax guidelines described
11224 in 2.10.2, except that the order of presentation of the −e and −f options is
11225 significant.

11226 The following options shall be supported by the implementation:

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

11227 `-e` *script* Add the editing commands specified by the *script* option-
11228 argument to the end of the script of editing commands. The *script*
11229 option-argument shall have the same properties as the *script*
11230 operand, described in 4.55.4.

11231 `-f` *script_file*
11232 Add the editing commands in the file *script_file* to the end of the
11233 script.

11234 `-n` Suppress the default output (in which each line, after it is exam-
11235 ined for editing, is written to standard output). Only lines expli-
11236 citly selected for output shall be written.

11237 Multiple `-e` and `-f` options may be specified. All commands shall be added to the
11238 script in the order specified, regardless of their origin.

11239 ### 4.55.4 Operands

11240 The following operands shall be supported by the implementation:

11241 *file* A pathname of a file whose contents shall be read and edited. If
11242 multiple *file* operands are specified, the named files shall be read
11243 in the order specified and the concatenation shall be edited. If no
11244 *file* operands are specified, the standard input shall be used.

11245 *script* A string to be used as the script of editing commands. The appli-
11246 cation shall not present a *script* that violates the restrictions of a
11247 text file (see 2.2.2.151), except that the final character need not be
11248 a `<newline>`.

11249 ### 4.55.5 External Influences

11250 #### 4.55.5.1 Standard Input

11251 The standard input shall be used only if no *file* operands are specified. See Input
11252 Files.

11253 #### 4.55.5.2 Input Files

11254 The input files shall be text files. The *script_file* s named by the `-f` option shall
11255 consist of editing commands, one per line.

11256 #### 4.55.5.3 Environment Variables

11257 The following environment variables shall affect the execution of `sed`:

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

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_COLLATE This variable shall determine the locale for the behavior of ranges, equivalence classes, and multicharacter collating elements within regular expressions.

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), and the behavior of character classes within regular expressions.

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

4.55.5.4 Asynchronous Events

Default.

4.55.6 External Effects

4.55.6.1 Standard Output

The input files shall be written to standard output, with the editing commands specified in the script applied. If the **−n** option is specified, only those input lines selected by the script shall be written to standard output.

4.55.6.2 Standard Error

Used only for diagnostic messages.

4.55.6.3 Output Files

The output files shall be text files whose formats are dependent on the editing commands given.

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

4.55.7 Extended Description

The *script* shall consist of editing commands, one per line, of the following form:

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

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

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

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

4.55.7.1 sed Addresses

An address is either empty, a decimal number that counts input lines cumulatively across files, a $ character that addresses the last line of input, or a context address (which consists of a regular expression as described in 4.55.7.2, preceded and followed by a delimiter, usually a slash).

A command line with no addresses shall select every pattern space.

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

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

Editing commands can be applied only to nonselected pattern spaces by use of the negation command ! (see 4.55.7.3).

4.55.7.2 sed Regular Expressions

The sed utility shall support the basic regular expressions described in 2.8.3, with the following additions:

(1) In a context address, the construction *cREc* , where *c* is any character other than <backslash> or <newline>, shall be identical to */RE/* . If the character designated by *c* appears following a backslash, then it shall be considered to be that literal character, which shall not terminate the RE. For example, in the context address \xabc\xdefx, the second x

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

stands for itself, so that the regular expression is `abcxdef`.

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

4.55.7.3 `sed` Editing Commands

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

The argument *text* shall consist of one or more lines. Each embedded `<newline>` in the text shall be preceded by a backslash. Other backslashes in text shall be removed and the following character shall be treated literally.

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

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

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

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

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

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

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

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

[*2addr*]`b` [*label*]
Branch to the `:` command bearing the *label* . If *label* is not specified, branch to the end of the script. The implementation

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

shall support *labels* recognized as unique up to at least 8 characters; the actual length (≥8) that shall be supported by the implementation is unspecified. It is unspecified whether exceeding a label length causes an error or a silent truncation.

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

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

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

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

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

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

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

[*1addr*]i\
text
Write *text* to standard output.

[*2addr*]l
(The letter ell.) Write the pattern space to standard output in a visually unambiguous form. The characters listed in Table 2-15 (see 2.12) shall be written as the corresponding escape sequence. Nonprintable characters not in Table 2-15 shall be written as one three-digit octal number (with a preceding <backslash>) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than nine bits, the format used for nonprintable characters is implementation defined.

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

[*2addr*]n
Write the pattern space to standard output if the default output has not been suppressed, and replace the pattern space with the next line of input.

[*2addr*]N
Append the next line of input to the pattern space, using an embedded <newline> to separate the appended material from the original material. Note that the current line number changes.

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

[*2addr*]p Write the pattern space to standard output.

[*2addr*]P Write the pattern space, up to the first <newline>, to standard output.

[*1addr*]q Branch to the end of the script and quit without starting a new cycle.

[*1addr*]r *rfile*
Copy the contents of *rfile* to standard output just before each attempt to fetch a line of input. If *rfile* does not exist or cannot be read, it shall be treated as if it were an empty file, causing no error condition.

[*2addr*]s/*regular expression* /*replacement* /*flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character other than <backslash> or <newline> can be used instead of a slash to delimit the RE and the replacement. Within the RE and the replacement, the RE delimiter itself can be used as a literal character if it is preceded by a backslash.

An ampersand (&) appearing in the *replacement* shall be replaced by the string matching the RE. The special meaning of & in this context can be suppressed by preceding it by backslash. The characters *n*, where *n* is a digit, shall be replaced by the text matched by the corresponding backreference expression (see 2.8.3.3).

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

The value of *flags* shall be zero or more of:

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

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

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

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

[*2addr*]t [*label*]
Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input

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

line or execution of a `t`. If *label* is not specified, branch to the end of the script.

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

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

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

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

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

[*1addr* **]=** Write the following to standard output:

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

[*0addr* **]** An empty command shall be ignored.

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

4.55.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

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

11480 ### 4.55.9 Consequences of Errors

11481 Default.

11482 ### 4.55.10 Rationale. *(This subclause is not a part of P1003.2)*

11483 **Examples, Usage**

11484 See the rationale for `cat` (4.4.10) for an example `sed` script.

11485 This standard requires implementations to support at least nine distinct *wfiles,*
11486 matching historical practice on many implementations. Implementations are
11487 encouraged to support more, but portable applications should not exceed this
11488 limit.

11489 Note that regular expressions match entire strings, not just individual lines, but
11490 `<newline>` is matched by `\n` in a `sed` RE; `<newline>` is not allowed in an RE.
11491 Also note that `\n` cannot be used to match a `<newline>` at the end of an input
11492 line; `<newline>`s appear in the pattern space as a result of the `N` editing com-
11493 mand.

11494 The exit status codes specified here are different from those in System V.
11495 System V returns 2 for garbled `sed` commands, but returns zero with its usage
11496 message or if the input file could not be opened. The working group considered
11497 this to be a bug.

11498 **History of Decisions Made**

11499 The manner in which the `l` command writes nonprintable characters was
11500 changed to avoid the historical backspace-overstrike method and added other 1
11501 requirements to achieve unambiguous output. See the rationale for `ed` (4.20.10) 1
11502 for details of the format chosen, which is the same as that chosen for `sed`. 1

11503 The standard requires implementations to provide pattern and hold spaces of at
11504 least 8192 bytes, larger than the 4000-byte spaces used by some historical imple-
11505 mentations, but less than the 20K byte limit used in an earlier draft. Implemen-
11506 tations are encouraged to dynamically allocate larger pattern and hold spaces as
11507 needed.

11508 The requirements for acceptance of `<blank>`s and `<space>`s in command lines
11509 has been made more explicit than in earlier drafts to clearly describe existing
11510 practice and remove confusion about the phrase ‘‘protect initial blanks [sic] and
11511 tabs from the stripping that is done on every script line’’ that appears in much of
11512 the historical documentation of the `sed` utility description of text. (Not all imple- 1
11513 mentations are known to have stripped `<blank>`s from text lines, although they 1
11514 all have allowed leading `<blank>`s preceding the address on a command line.) 1

11515 The treatment of # comments differs from the *SVID* , which only allows a comment
11516 as the first line of the script, but matches BSD-derived implementations. The
11517 comment character is treated as a command and it has the same properties in
11518 terms of being accepted with leading `<blank>`s; the BSD implementation has

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

historically supported this.

Earlier drafts of POSIX.2 required that a *script_file* have at least one noncomment line. Some historical implementations have behaved in unexpected ways if this were not the case. The working group felt that this was incorrect behavior, and that application developers should not have to work around this feature. A correct implementation of POSIX.2 shall permit *script_file* s that consist only of comment lines.

Earlier drafts indicated that if `-e` and `-f` options were intermixed, all `-e` options were processed before any `-f` options. This has been changed to process them in the order presented because it matches existing practice and is more intuitive.

The treatment of the `p` flag to the `s` command differs between System V and BSD-based systems (actually, between Version 7 and 32V) when the default output is suppressed. In the two examples:

```
echo a | sed    's/a/A/p'
echo a | sed -n 's/a/A/p'
```

POSIX.2, BSD, System V documentation, and the *SVID* indicate that the first example should write two lines with `A`, whereas the second should write one. Some System V systems write the `A` only once in both examples, because the `p` flag is ignored if the `-n` option is not specified.

This is a case of a diametrical difference between systems that could not be reconciled through the compromise of declaring the behavior to be unspecified. The *SVID*/BSD/32V behavior was adopted for POSIX.2 because:

— No known documentation for any historic system describes the interaction between the `p` flag and the `-n` option.

— The selected behavior is more correct as there is no technical justification for any interaction between the `p` flag and the `-n` option. A relationship between `-n` and the `p` flag might imply that they are only used together (when `p` should be a no-op), but this ignores valid scripts that interrupt the cyclical nature of the processing through the use of the `D`, `d`, `q`, or branching commands. Such scripts rely on the `p` suffix to write the pattern space because they do not make use of the default output at the ‘‘bottom’’ of the script.

— Because the `-n` option makes the `p` flag a no-op, any interaction would only be useful if `sed` scripts were written to run both with and without the `-n` option. This is believed to be unlikely. It is even more unlikely that programmers have coded the `p` flag expecting it to be a no-op. Because the interaction was not documented, the likelihood of a programmer discovering the interaction and depending on it is further decreased.

— Finally, scripts that break under the specified behavior will produce too much output instead of too little, which is easier to diagnose and correct.

The form of the substitute command that uses the *n* suffix was limited to the first 512 matches in a previous draft. This limit has been removed because there is no reason an editor processing lines of {LINE_MAX} length should have this

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

restriction. The command `s/a/A/2047` should be able to substitute the 2047th occurrence of `a` on a line.

4.56 `sh` — Shell, the standard command language interpreter

4.56.1 Synopsis

`sh [-aCefinuvx][` *command_file* `[`*argument* `...]]`

`sh -c [-aCefinuvx]` *command_string* `[` *command_name* `[`*argument* `...]]`

`sh -s [-aCefinuvx][`*argument* `...]`

4.56.2 Description

The `sh` utility is a command language interpreter that shall execute commands read from a command-line string, the standard input, or a specified file. The commands to be executed shall be expressed in the language described in Section 3.

4.56.3 Options

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

The `-a`, `-C`, `-e`, `-f`, `-n`, `-u`, `-v`, and `-x` options are described as part of the `set` utility in 3.14.11. The following additional options shall be supported by the implementation:

`-c` Read commands from the *command_string* operand. Set the value of special parameter `0` (see 3.5.2) from the value of the *command_name* operand and the positional parameters (`$1`, `$2`, etc.) in sequence from the remaining *argument* operands. No commands shall be read from the standard input.

`-i` Specify that the shell is *interactive* ; see below. An implementation may treat specifying the `-i` option as an error if the real user ID of the calling process does not equal the effective user ID or if the real group ID does not equal the effective user ID.

`-s` Read commands from the standard input.

If there are no operands and the `-c` option is not specified, the `-s` option shall be assumed.

If the `-i` option is present, or if there are no operands and the shell's standard input and standard error are attached to a terminal, the shell is considered to be *interactive* . (See 3.1.4.) The behavior of an interactive shell is not fully specified by this standard.

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

NOTE: The preceding sentence is expected to change following the eventual approval of the UPE supplement.

Implementations may accept the option letters with a leading plus sign (+) instead of a leading hyphen (meaning the reverse case of the option as described in this standard). A conforming application shall protect its first operand, if it starts with a plus sign, by preceding it with the −− argument that denotes ‘‘end of options.’’

4.56.4 Operands

The following operands shall be supported by the implementation:

`-` A single hyphen shall be treated as the first operand and then ignored. If both `-` and `--` are given as arguments, or if other operands precede the single hyphen, the results are undefined.

argument The positional parameters (`$1`, `$2`, etc.) shall be set to *arguments* , if any.

command_file
The pathname of a file containing commands. If the pathname contains one or more slash characters, the implementation shall attempt to read that file; the file need not be executable. If the pathname does not contain a slash character:

— The implementation shall attempt to read that file from the current working directory; the file need not be executable.

— If the file is not in the current working directory, the implementation may perform a search for an executable file using the value of **PATH** , as described in 3.9.1.1.

Special parameter `0` (see 3.5.2) shall be set to the value of *command_file* . If `sh` is called using a synopsis form that omits *command_file* , special parameter `0` shall be set to the value of the first argument passed to `sh` from its parent (e.g., *argv* [0] in the C binding), which is normally a pathname used to execute the `sh` utility.

command_name
A string assigned to special parameter `0` when executing the commands in *command_string*. If *command_name* is not specified, special parameter `0` shall be set to the value of the first argument passed to `sh` from its parent (e.g., *argv* [0] in the C binding), which is normally a pathname used to execute the `sh` utility.

command_string
A string that shall be interpreted by the shell as one or more commands, as if the string were the argument to the function in 7.1.1 [such as the *system* () function in the C binding]. If the *command_string* operand is an empty string, `sh` shall exit with a

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

zero exit status.

4.56.5 External Influences

4.56.5.1 Standard Input

The standard input shall be used only if:

(1) The `-s` option is specified, or;

(2) The `-c` option is not specified and no operands are specified, or;

(3) The script executes one or more commands that require input from standard input (such as a `read` command that does not redirect its input).

See Input Files.

When the shell is using standard input and it invokes a command that also uses standard input, the shell shall ensure that the standard input file pointer points directly after the command it has read when the command begins execution. It shall not read ahead in such a manner that any characters intended to be read by the invoked command are consumed by the shell (whether interpreted by the shell or not) or that characters that are not read by the invoked command are not seen by the shell. When the command expecting to read standard input is started asynchronously by an interactive shell, it is unspecified whether characters are read by the command or interpreted by the shell.

If the standard input to `sh` is a FIFO or terminal device and is set to nonblocking reads, then `sh` shall enable blocking reads on standard input. This shall remain in effect when the command completes.

4.56.5.2 Input Files

The input file shall be a text file, except that line lengths shall be unlimited. If the input file is empty or consists solely of blank lines and/or comments, `sh` shall exit with a zero exit status.

4.56.5.3 Environment Variables

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

HOME This variable shall be interpreted as the pathname of the user's home directory. The contents of **HOME** are used in Tilde Expansion as described in 3.6.1.

IFS *Input field separators* : a string treated as a list of characters that shall be used for field splitting and to split lines into words with the `read` command. See 3.6.5. If **IFS** is not set, the shell shall behave as if the value of **IFS** were the `<space>`, `<tab>`, and `<newline>` characters. Implementations may ignore the value of **IFS** in the

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

environment at the time `sh` is invoked, treating **IFS** as if it were not set.

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_COLLATE This variable shall determine the behavior of range expressions, equivalence classes, and multicharacter collating elements within pattern matching.

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), which characters are defined as letters (character class `alpha`), and the behavior of character classes within pattern matching.

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

PATH This variable shall represent a string formatted as described in 2.6, used to effect command interpretation. See 3.9.1.1.

4.56.5.4 Asynchronous Events

Default.

4.56.6 External Effects

4.56.6.1 Standard Output

See Standard Error.

4.56.6.2 Standard Error

Except as otherwise stated (by the descriptions of any invoked utilities or in interactive mode), standard error is used only for diagnostic messages.

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