-- --

IEEE P1003.2 Draft 11.2 – September 1991

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

Examples, Usage

On historical System V systems, the `−p` option also caused a message to be written to the standard output. The message indicated whether the whole path was removed or part of the path remains for some reason. The Standard Error subclause requires this diagnostic when the entire path specified by a *dir* operand is not removed, but does not allow the status message reporting success to be written as a diagnostic.

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

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

will remove all three directories.

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

4.55 `sed` — Stream editor

4.55.1 Synopsis

`sed [−n]` *script* [*file* . . .]

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

4.55.2 Description

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

4.55.3 Options

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

The following options shall be supported by the implementation:

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

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

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

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

4.55.4 Operands

The following operands shall be supported by the implementation:

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

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

4.55.5 External Influences

4.55.5.1 Standard Input

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

4.55.5.2 Input Files

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

4.55.5.3 Environment Variables

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

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.

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`

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. 1

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

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

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

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

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

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

`[`*2addr*`]G` — Append to the pattern space a `<newline>` followed by the con- 1
tents of the hold space. 1

`[`*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 1
of the pattern space. 1

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

`[`*2addr*`]l` — (The letter ell.) Write the pattern space to standard output in a 1
visually unambiguous form. The characters listed in Table 2-15 1
(see 2.12) shall be written as the corresponding escape sequence. 1
Nonprintable characters not in Table 2-15 shall be written as one 1
three-digit octal number (with a preceding `<backslash>`) for 1
each byte in the character (most significant byte first). If the size 1
of a byte on the system is greater than nine bits, the format used 1
for nonprintable characters is implementation defined. 1

Long lines shall be folded, with the point of folding indicated by 1
writing `<backslash><newline>`; the length at which folding 1
occurs is unspecified, but should be appropriate for the output 1
device. The end of each line shall be marked with a `$`. 1

`[`*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.

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

`[`*2addr*`]P` Write the pattern space, up to the first `<newline>`, to standard 1
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 1
attempt to fetch a line of input. If *rfile* does not exist or cannot be 1
read, it shall be treated as if it were an empty file, causing no 1
error condition. 1

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

Substitute the *replacement* string for instances of the *regular*
expression in the pattern space. Any character other than 1
`<backslash>` or `<newline>` can be used instead of a slash to 1
delimit the RE and the replacement. Within the RE and the 1
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. 1
The application shall escape the `<newline>` in the *replacement* 1
by preceding it by backslash. A substitution shall be considered 1
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 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 1
instead of slash to delimit the strings. Within *string1* and 1
string2, the delimiter itself can be used as a literal character if it 1
is preceded by a backslash. 1

[*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*> 1

[*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.

4.55.9 Consequences of Errors

Default.

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

Examples, Usage

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

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

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

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

History of Decisions Made

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

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

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

The treatment of # comments differs from the *SVID*, which only allows a comment as the first line of the script, but matches BSD-derived implementations. The comment character is treated as a command and it has the same properties in terms of being accepted with leading `<blank>`s; the BSD implementation has 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 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* ...]] 1

`sh −c` [−aCefinuvx] *command_string* [*command_name* [*argument* ...]] 1

`sh −s` [−aCefinuvx][*argument* ...] 1

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.