IEEE P1003.2a Draft 8 – December 1991

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

None.

4.56 sh — Shell, the standard command language interpreter

⇒ **4.56.1 sh Synopsis.** *Add the* –b, –m, *and* –o *option options to each of the
three Synopsis lines with appropriate editorial wording indicating that these* 7
options need to be supported only when the UPUO is supported: 7

⇒ **4.56.3 sh Options.** *Change the first sentence of the second paragraph, beginning ‘The –a, –C, . . . ’’to:*

The –a, –b, –C, –e, –f, –m, –n, –o *option* , –u, –v, and –x options are described as part of the set utility in 3.14.11.

⇒ **4.56.3 sh Options.** *Remove the following sentence and NOTE:* 8

The behavior of an interactive shell is not fully specified by this standard. 8

NOTE: The preceding sentence is expected to change following the eventual approval of the UPE 8
supplement. 8

⇒ **4.56.5.3 sh Environment Variables.** *Modify the contents of subclause 4.56.5.3, Environment Variables, to add the following environment variables in the correct sorted order.*

FCEDIT 8
HISTFILE 8
HISTSIZE See the description of the fc environment variables in 8
5.12.5.3. This standard specifies the effects of these 8
variables only for systems supporting the User Porta- 8
bility Utilities Option. 8

MAIL This variable shall be interpreted as a pathname of the user's mailbox file for purposes of incoming mail notification. If this variable is set, the shell shall inform the user if the file named by the variable is created or if its modification time has changed. Informing the user shall be accomplished by writing a string of unspecified format to the standard error prior to the writing of the next primary prompt string after the completion of an interval defined by the **MAIL-CHECK** variable. The user shall be informed only if
MAIL is set and **MAILPATH** is not set. This standard 7

specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

MAILCHECK This variable shall be interpreted as a decimal integer value that specifies how often (in seconds) the shell shall check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** variables. The default value shall be 600 seconds. If set to zero, the shell shall check before issuing each primary prompt. This standard specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

MAILPATH This variable shall be interpreted as a list of pathnames and optional messages separated by colons. If this variable is set, the shell shall inform the user if any of the files named by the variable are created or if any of their modification times change. (See the preceding entry for **MAIL** for descriptions of mail arrival and user informing.) Each pathname can be followed by `%` and a string that shall be subjected to parameter expansion and written to standard error when the modification time changes. If a `%` character in the pathname is preceded by a backslash, it shall be treated as a literal `%` in the pathname. The default message is unspecified. This standard specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

The **MAILPATH** environment variable shall take precedence over the **MAIL** variable.

⇒ **4.56.7 `sh` Extended Description.** *Replace the subclause 4.56.7, Extended Description, with the following.*

4.56.7 Extended Description

See the description of the shell command language in Section 3.

On systems supporting the User Portability Utilities Option, the following additional capabilities shall be supported.

4.56.7.1 Command History List

When the `sh` utility is being used interactively, it shall maintain a list of commands previously entered from the terminal in the file named by the **HISTFILE** environment variable. The type, size, and internal format of this file are unspecified. Multiple `sh` processes can share access to the file for a user, if file access permissions allow this; see the description of the **HISTFILE** environment variable in 5.12.5.3.

4.56.7.2 Command Line Editing

When `sh` is being used interactively from a terminal, the current command and the command history (see `fc` in 5.12) can be edited using *vi-mode* command-line editing. This mode uses commands, described below, similar to a subset of those described in the `vi` (see 5.35) utility. Implementations may offer other command-line editing modes corresponding to other editing utilities.

The command `set -o vi` shall enable vi-mode editing and place `sh` into `vi` insert mode (see 4.56.7.3). This command shall also disable any other editing mode that the implementation may provide. The command `set +o vi` shall disable vi-mode editing.

Certain block-mode terminals may be unable to support shell command line editing. If a terminal is unable to provide either edit mode, it need not be possible to `set -o vi` when using the shell on this terminal.

In the following subclauses, the characters *erase*, *interrupt*, *kill*, and *end-of-file* are those set by the `stty` utility; see 4.59.

4.56.7.3 Vi-Mode Command Line Editing

With vi-mode enabled, `sh` can be switched between insert mode and command mode.

When in insert mode, an entered character shall be inserted into the command line, except as noted in 4.56.7.4. Upon entering `sh` and after termination of the previous command, `sh` shall be in insert mode.

Typing <ESC> shall switch `sh` into command mode (see 4.56.7.5). In command mode, an entered character shall either invoke a defined operation, be used as part of a multicharacter operation, or be treated as an error. A character that is not recognized as part of an editing command shall terminate any specific editing command and shall alert the terminal. Typing the *interrupt* character in command mode shall cause `sh` to terminate command-line editing on the current command line, re-issue the prompt on the next line of the terminal, and reset the command history (see 5.12) so that the most recently executed command is the ‘‘previous’’ command (i.e., the command that was being edited when it was interrupted is not re-entered into the history).

In the following subclauses, the phrase ‘‘move the cursor to the beginning of the word’’ shall mean ‘‘move the cursor to the first character of the current word’’ and the phrase ‘‘move the cursor to the end of the word’’ shall mean ‘‘move the cursor to the last character of the current word.’’ The phrase ‘‘beginning of the command line’’ indicates the point between the end of the prompt string issued by the shell (or the beginning of the terminal line, if there is no prompt string) and the first character of the command text.

4.56.7.4 Vi Line Editing Insert Mode

While in insert mode, any character typed shall be inserted in the current command line, unless it is from the following set.

`<newline>` Execute the current command line being edited. 7

erase Delete the character previous to the current cursor position and move the current cursor position back one character. In insert mode, characters shall be erased from both the screen and the buffer when backspacing.

interrupt Terminate command-line editing with the same effects as 8
described for interrupting command mode; see 4.56.7.3. 8

kill Clear all the characters from the input line.

`control-V` Insert the next character input, even if the character is otherwise a special insert mode character.

`control-W` Delete the characters from the one preceding the cursor to the
preceding word boundary. The word boundary in this case is the 7
closer to the cursor of either the beginning of the line or a char- 7
acter that is in neither the `blank` nor `punct` character 7
classification of the current locale. 7

`\` If followed by the *erase* or *kill* character, that character shall be inserted into the input line. Otherwise, the backslash itself shall be inserted into the input line.

end-of-file Interpreted as the end of input in `sh`. This interpretation shall occur only at the beginning of an input line. If end-of-file is entered other than at the beginning of the line, the results are unspecified.

`<ESC>` Place `sh` into command mode.

4.56.7.5 Vi Line Editing Command Mode

In command mode for the command-line editing feature, decimal digits not beginning with 0 that precede a command letter shall be remembered. Some commands use these decimal digits as a count number that affects the operation.

The term *motion command* in the following list represents one of the commands:

`<space>`	`0`	`b`	`F`	`l`	`W`	`^`
`$`	`;`	`E`	`f`	`T`	`w`	`|`
`,`	`B`	`e`	`h`	`t`		

Any command that modifies the current line shall cause a copy of the current line 7
to be made at the end of the command history, the current line shall become that 8
copy, and the edit shall be performed on that copy. 7

Any command that is preceded by *count* shall take a count (the numeric value of any preceding decimal digits). Unless otherwise noted, this count shall cause the specified operation to repeat by the number of times specified by the count. Also unless otherwise noted, a *count* that is out of range is considered an error condition and shall alert the terminal, but neither the cursor position, nor the command line, shall change.

The terms *word* and *bigword* are used as defined in the `vi` description; see
5.35.7.1. The term *save buffer* corresponds to the term *unnamed buffer* in 7
5.35.7.1. 7

The following commands shall be recognized in command mode:

`<newline>` Execute the current command line being edited.

`control-L` Redraw the current command line. Position the cursor at the same location on the new command line.

`#` Insert the character # at the beginning of the current command
line and treat the current command line as a comment. This 8
line shall be entered into the command history; see 5.12. 8

`=` Display the possible shell word expansions (see 3.6) of the bigword at the current command-line position. These expansions shall be displayed on subsequent terminal lines. If the bigword contains none of the characters `?`, `*`, or `[`, an asterisk (`*`) shall be implicitly assumed at the end. If any directories are matched, these expansions shall have a `/` character appended. After the expansion, the line shall be redrawn, the cursor shall be repositioned at the current cursor position, and `sh` shall be placed in command mode.

`\` Perform pathname expansion (see 3.6.6) on the current bigword, up to the largest set of characters that can be matched uniquely. If the bigword contains none of the characters `?`, `*`, or `[`, an asterisk (`*`) shall be implicitly assumed at the end. This maximal expansion then shall replace the original bigword in the command line, and the cursor shall be placed after this expansion. If the resulting bigword completely and uniquely matches a directory, a `/` character shall be inserted directly after the bigword. If some other file is completely matched, a single `<space>` character shall be inserted after the bigword. After this operation, `sh` shall be placed in insert mode.

`*` Perform pathname expansion on the current bigword and insert all expansions into the command to replace the current bigword, with each expansion separated by a single `<space>` character. If at the end of the line, the current cursor position shall be moved to the first column position following the expansions and `sh` shall be placed in insert mode. Otherwise, the current cursor position shall be the last column position of the first character after the expansions and `sh` shall be placed in insert mode. If the current bigword contains none of the characters `?`, `*`, or `[`, before the operation, an asterisk shall be implicitly assumed at the end.

@letter
Insert the value of the alias named *_letter*. The symbol *letter* 8
represents a single alphabetic character from the portable char- 8
acter set; implementations may support additional characters as 8
an extension. If the alias *letter* contains other editing com- 8
mands, these commands shall be performed as part of the insertion. If no alias *letter* is enabled, this command shall have no effect.

[*count*]~
Convert, if the current character is a lowercase letter, to the equivalent uppercase letter, and vice versa, as prescribed by the current locale. The current cursor position then shall be advanced by one character. If the cursor was positioned on the last character of the line, the case conversion shall occur, but the cursor shall not advance. If the ~ command is preceded by a
count, that number of characters shall be converted, and the cur- 7
sor shall be advanced to the character position after the last 7
character converted. If the *count* is larger than the number of 7
characters after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]`.`
Repeat the most recent nonmotion command, even if it was exe- 7
cuted on an earlier command line. If the previous command was 7
preceded by a *count*, and no count is given on the `.` command, the count from the previous command shall be included as part of the repeated command. If the `.` command is preceded by a *count*, this shall override any *count* argument to the previous
command. The *count* specified in the `.` command shall become 7
the count for subsequent `.` commands issued without a count. 7

[*number*]`v`
Invoke the `vi` editor to edit the current command line in a temporary file. When the editor exits, the commands in the temporary file shall be executed. If a *number* is prefixed to the com-
mand, it specifies the command number in the command history 8
to be edited, rather than the current command line.

[*count*]`l` (ell)
[*count*]`<space>`
Move the current cursor position to the next character position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of characters after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]`h`
Move the current cursor position to the *count*-th (default 1) previous character position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the count is larger than the number of characters before the cursor, this shall not be considered an

error; the cursor shall move to the first character on the line.

[*count*]w
: Move to the start of the next word. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of words after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]W
: Move to the start of the next bigword. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of bigwords after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]e
: Move to the end of the current word. If at the end of a word, move to the end of the next word. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of words after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]E
: Move to the end of the current bigword. If at the end of a bigword, move to the end of the next bigword. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of bigwords after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]b
: Move to the beginning of the current word. If at the beginning of a word, move to the beginning of the previous word. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the *count* is larger than the number of words preceding the cursor, this shall not be considered an error; the cursor shall return to the first character on the line.

[*count*]B
: Move to the beginning of the current bigword. If at the beginning of a bigword, move to the beginning of the previous bigword. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the *count* is larger than the number of bigwords preceding the cursor, this shall not be considered an error; the cursor shall return to the first character on the line.

^
: Move the current cursor position to the first character on the input line that is not a <blank>.

`$` Move to the last character position on the current command line.

`0` (Zero.) Move to the first character position on the current command line.

[*count*]`|` Move to the *count*-th character position on the current command line. If no number is specified, move to the first position. The first character position shall be numbered 1. If the count is larger than the number of characters on the line, this shall not be considered an error; the cursor shall be placed on the last character on the line.

[*count*]`f`*c* Move to the first occurrence of the character *c* that occurs after the current cursor position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the character *c* does not occur in the line after the current cursor position, the terminal shall be alerted and the cursor shall not be moved.

[*count*]`F`*c* Move to the first occurrence of the character *c* that occurs before the current cursor position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the character *c* does not occur in the line before the current cursor position, the terminal shall be alerted and the cursor shall not be moved.

[*count*]`t`*c* Move to the character before the first occurrence of the character *c* that occurs after the current cursor position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the character *c* does not occur in the line after the current cursor position, the terminal shall be alerted and the cursor shall not be moved.

[*count*]`T`*c* Move to the character after the first occurrence of the character *c* that occurs before the current cursor position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the character *c* does not occur in the line before the current cursor position, the terminal shall be alerted and the cursor shall not be moved.

[*count*]`;` Repeat the most recent `f`, `F`, `t`, or `T` command. Any number argument on that previous command shall be ignored. Errors are those described for the repeated command.

[*count*]`,` Repeat the most recent `f`, `F`, `t`, or `T` command, ignoring the number argument to that command. However, reverse the direction of that command.

`a` Enter insert mode after the current cursor position. Characters that are entered shall be inserted before the next character.

`A` Enter insert mode after the end of the current command line.

`i` Enter insert mode at the current cursor position. Characters that are entered shall be inserted before the current character.

`I` Enter insert mode at the beginning of the current command line.

`R` Enter insert mode, replacing characters from the command line beginning at the current cursor position.

`[`*count*`]c`*motion*

Delete the characters between the current cursor position and the cursor position that would result from the specified *motion* command. Then enter insert mode before the first character following any deleted characters. If *count* is specified, it shall be applied to the *motion* command. A *count* shall be ignored for the motion commands:

`0` `^` `$` `c`

If the *motion* command is the character `c`, the current command line shall be cleared and insert mode shall be entered. If the *motion* command would move the current cursor position toward the beginning of the command line, the character under the current cursor position shall not be deleted. If the motion command would move the current cursor position toward the end of the command line, the character under the current cursor position shall be deleted. If the *count* is larger than the number of characters between the current cursor position and the end of the command line toward which the motion command would move the cursor, this shall not be considered an error; all of the remaining characters in the aforementioned range shall be deleted and insert mode shall be entered. If the motion command is invalid, the terminal shall be alerted, the cursor shall not be moved, and no text shall be deleted.

`C` Delete from the current character to the end of the line and enter insert mode at the new end of line.

`S` Clear the entire current command line and enter insert mode.

`[`*count*`]r`*c* Replace the current character with the character *c*. With a number *count*, replace the current and the following *count* −1 characters. After this command, the current cursor position shall be on the last character that was changed. If the *count* is larger than the number of characters after the cursor, this shall not be considered an error; all of the remaining characters shall be changed.

`[`*count*`]_` Append a `<space>` after the current character position and then append the last bigword in the previous input line after the `<space>`. Then enter insert mode after the last character just appended. With a number *count*, append the *count*-th bigword in the previous line.

[*count*]`x` Delete the character at the current cursor position and place the deleted character(s) in the save buffer. If the cursor was positioned on the last character of the line, the character shall be deleted and the cursor position shall be moved to the previous character (the new last character). If the *count* is larger than the number of characters after the cursor, this shall not be considered an error; all the characters from the cursor to the end of the line shall be deleted.

[*count*]`X` Delete the character before the current cursor position and place 7
the deleted character(s) in the save buffer. The character under 7
the current cursor position shall not change. If the cursor was positioned on the first character of the line, the terminal shall be alerted, and the `X` command shall have no effect. If the line contained a single character, the `X` command shall have no effect. If the line contained no characters, the terminal shall be alerted and the cursor shall not be moved. If the *count* is larger than the number of characters before the cursor, this shall not be considered an error; all the characters from before the cursor to the beginning of the line shall be deleted.

[*count*]`d` *motion*

Delete the characters between the current cursor position and the character position that would result from the *motion* command. A number *count* repeats the *motion* command *count* times. If the *motion* command would move toward the beginning of the command line, the character under the current cursor position shall not be deleted. If the *motion* command is `d`, the entire current command line shall be cleared. If the *count* is larger than the number of characters between the current cursor position and the end of the command line toward which the motion command would move the cursor, this shall not be considered an error; all of the remaining characters in the aforementioned range shall be deleted. The deleted characters shall be placed in the save buffer.

`D` Delete all characters from the current cursor position to the end of the line. The deleted characters shall be placed in the save buffer.

[*count*]`y` *motion*

Yank (i.e., copy) the characters from the current cursor position to the position resulting from the *motion* command into a save buffer. A number *count* shall be applied to the *motion* command. If the *motion* command would move toward the beginning of the command line, the character under the current cursor position shall not be included in the set of yanked characters. If the *motion* command is `y`, the entire current command line shall be yanked into the save buffer. The current cursor position shall be unchanged. If the *count* is larger than the number of characters between the current cursor position and the end of the command line toward which the motion command would move the cursor, this shall not be considered an error; all of the remaining characters in the aforementioned range shall be yanked.

`Y` Yank the characters from the current cursor position to the end of the line into the save buffer. The current character position shall be unchanged.

[*count*]`p` Put a copy of the current contents of the save buffer after the current cursor position. The current cursor position shall be advanced to the last character put from the save buffer. A *count* shall indicate how many copies of the save buffer shall be put.

[*count*]`P` Put a copy of the current contents of the save buffer before the current cursor position. The current cursor position shall be moved to the last character put from the save buffer. A *count* shall indicate how many copies of the save buffer shall be put.

`u` Undo the last command that modified the text of the current command line.

`U` Undo all changes made to the current command line since first entering command mode on the line.

[*count*]`k`
[*count*]`–` Replace the current command line with the previous command line in the shell command history. The cursor shall be positioned on the first character of the new command. A count preceding the command shall have the same effect as executing the command *count* times. If a `k` or `-` command retreats past the maximum number of commands in effect for this shell (affected by the **HISTSIZE** environment variable), the terminal shall be alerted and the command shall have no effect.

[*count*]`j`
[*count*]`+` Replace the current command line with the next command line in the shell command history. The cursor shall be positioned on the first character of the new command. The command history 8
position shall be remembered, and any `k` or `–` command, or `j` or `+` command, shall decrement or increment that position and then shall fetch the line at the new position. If a `j` or `+` command advances past the most recent line in the history, the current command line shall be restored to the contents before the first `k` or `–`. 7

[*number*]`G` Replace the current command line with the contents of the oldest command line stored in the shell command history. With a number *number*, replace the current command line with the contents of command *number* in the history.

`/`*string* `<newline>`
Move backward through the command history, searching for the

specified *string*, beginning with the previous command line. If it is not found, the current command line shall be unchanged. If it is found in a previous line, this command shall behave equivalently to a set of `k` commands to reach that line. If *string* begins with ^, the characters after the ^ shall be matched only at the beginning of a line.

`?`*string* `<newline>`
: Move forward through the command history, searching for the specified string. If it is not found, the current command line shall be unchanged. If the string is found in the current command line, the current cursor position shall be moved to the beginning of that string. If it is found in the history, this command shall behave equivalently to a set of `j` commands to reach that line. If *string* begins with ^, the characters after the ^ shall be matched only at the beginning of a line.

`n`
: Repeat the most recent `/` or `?` command.

`N`
: Repeat the most recent `/` or `?` command, reversing the direction of the search.

8

4.56.10 `sh` Rationale. *(This subclause is not a part of P1003.2a)*

The default messages for the various **MAIL**-related messages are unspecified because they vary across implementations. Typical messages are:

`you have mail\n` or `you have new mail\n` 7

It is important that the descriptions of command line editing refer to the same shell as that in the base standard so that interactive users can also be application programmers without having to deal with programmatic differences in their two environments. It is also essential that the utility name `sh` be specified because this explicit utility name is too firmly rooted in historical practice of application programs for it to change.

Consideration was given to mandating a diagnostic message when attempting to 7
set vi-mode on terminals that do not support command line editing. However, it 8
is not historical practice for the shell to be cognizant of all terminal types and 7
thus be able to detect inappropriate terminals in all cases. Implementations are 7
encouraged to supply diagnostics in this case whenever possible, rather than leav- 7
ing the user in a state where editing commands work incorrectly. 7

In earlier drafts, the KornShell-derived `emacs` mode of command-line editing was 8
included, even though the `emacs` editor itself was not. The community of `emacs` 8
editing enthusiasts was quite adamant that the full `emacs` editor not be included in POSIX.2, because it was concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to the standard, but lacking the extensibility required by the community.

The author of the original `emacs` program also expressed his desire to omit the program. Furthermore, there were a number of historical systems that did not include `emacs`, or included it without supporting it, but there were very few that did not include and support `vi`. The shell `emacs` command-line editing mode was finally omitted from this standard because it became apparent that the KornShell version and the editor being distributed with the GNU system had diverged in some respects. The author of `emacs` requested that the POSIX `emacs` mode either be deleted or have a significant number of unspecified conditions. Although the KornShell author agreed to consider changes to bring the shell into alignment, the developers of the standard decided to defer specification at this time, rather than attempting to agree on a specific subset of `emacs` late within balloting of the standard. (Furthermore, the standard developers included few individuals who had direct experience with `emacs` mode and the specification did not achieve the level of review afforded to the `vi` mode.) It is assumed that the `emacs` and KornShell developers will converge on a definition acceptable to both groups and this may be used as a model for a future version of this standard. In the interim, implementations are free to offer additional command-line editing modes based on the exact models of editors their users are most comfortable with.

4.70 `wait` — Await Process Completion

⇒ **4.70 `wait` Operands.** *In subclause 4.70, modify the description of the pid operand in the Operands subclause:*

pid One of the following:

(1) *<current contents>*

(2) On systems supporting both the POSIX.1 {8} job control option and the User Portability Utilities Option, a job control job ID (see 2.2.2.206) that identifies a background process group to be waited for. The job control job ID notation is applicable only for invocations of `wait` in the current shell execution environment; see 3.12. The exit status of `wait` shall be determined by the last command in the pipeline.

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

This additional syntax for *pid* is applicable only to systems having the job control features of the User Portability Extension. Thus, it is described in this supplement document, rather than in the base POSIX.2 document. See the rationale for `kill` for additional information on the effects of the job control job ID notation outside the current shell execution environment.