-- --

IEEE P1003.2 Draft 11.2 – September 1991

4.29.6 External Effects

4.29.6.1 Standard Output

The standard output shall contain designated portions of the input file(s).

If multiple *file* operands are specified, `head` shall precede the output for each with the header:

`"\n==> %s <==\n"`, *<pathname>*

except that the first header written shall not include the initial `<newline>`.

4.29.6.2 Standard Error

Used only for diagnostic messages.

4.29.6.3 Output Files

None.

4.29.7 Extended Description

None.

4.29.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

4.29.9 Consequences of Errors

Default.

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

Usage, Examples

The nonobsolescent version of `head` was created to allow conformance to the Utility Syntax Guidelines. The `-n` option was added to this new interface so that `head` and `tail` would be more logically related.

To write the first ten lines of all files (except those with a leading period) in the directory:

```
head *
```

History of Decisions Made

The `head` utility was not in early drafts. It was felt that `head`, and its frequent companion, `tail`, were useful mostly to interactive users, and not application programs. However, balloting input suggested that these utilities actually do find significant use in scripts, such as to write out portions of log files. Although it is possible to simulate `head` with `sed 10q` for a single file, the working group decided that the popularity of `head` on historical BSD systems warranted its inclusion alongside `tail`.

An earlier draft had the synopsis line:

head [−c | −l] [−n *number*] [*file* . . .]

This was changed to the current form based on comments and objections noting that `−c` has not been provided by historical versions of `head` and other utilities in POSIX.2 provide similar functionality. Also, `−l` was changed to `−n` to match a similar change in `tail`.

4.30 id — Return user identity

4.30.1 Synopsis

id [*user*]

id −G [−n] [*user*]

id −g [−nr] [*user*]

id −u [−nr] [*user*]

4.30.2 Description

If no *user* operand is provided, the `id` utility shall write the user and group IDs and the corresponding user and group names of the invoking process to standard output. If the effective and real IDs do not match, both shall be written. If multiple groups are supported by the underlying system (see the description of {NGROUPS_MAX} in POSIX.1 {8}), the supplementary group affiliations of the invoking process also shall be written.

If a *user* operand is provided and the process has the appropriate privileges, the user and group IDs of the selected user shall be written. In this case, effective IDs
shall be assumed to be identical to real IDs. If the selected user has more than 1
one allowable group membership listed in the group database (see POSIX.1 {8} sec- 1
tion 9.1), these shall be written in the same manner as the supplementary groups 1
described in the preceding paragraph. 1

4.30.3 Options

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

The following options shall be supported by the implementation:

`−G` Output all different group IDs (effective, real, and supplementary) only, using the format `"%u\n"`. If there is more than one distinct group affiliation, output each such affiliation, using the format `" %u"`, before the `<newline>` is output.

`−g` Output only the effective group ID, using the format `"%u\n"`.

`−n` Output the name in the format `"%s"` instead of the numeric ID using the format `"%u"`.

`−r` Output the real ID instead of the effective ID.

`−u` Output only the effective user ID, using the format `"%u\n"`.

4.30.4 Operands

The following operand shall be supported by the implementation:

user The login name for which information is to be written.

4.30.5 External Influences

4.30.5.1 Standard Input

None.

4.30.5.2 Input Files

None.

4.30.5.3 Environment Variables

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

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

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

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

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

4.30.5.4 Asynchronous Events

Default.

4.30.6 External Effects

4.30.6.1 Standard Output

The following formats shall be used when the LC_MESSAGES locale category specifies the POSIX Locale. In other locales, the strings `uid`, `gid`, `euid`, `egid`, and `groups` may be replaced with more appropriate strings corresponding to the locale.

`"uid=%u(%s) gid=%u(%s)\n"`, *<real user ID>*, *<user-name>* ,
<real group ID>, *<group-name>*

If the effective and real user IDs do not match, the following shall be inserted immediately before the `\n` character in the previous format:

`" euid=%u(%s)"`,

with the following arguments added at the end of the argument list:

<effective user ID>, *<effective user-name>*

If the effective and real group IDs do not match, the following shall be inserted directly before the `\n` character in the format string (and after any addition resulting from the effective and real user IDs not matching):

`" egid=%u(%s)"`,

with the following arguments added at the end of the argument list:

<effective group-ID>, *<effective group name>*

If the process has supplementary group affiliations or the selected user is allowed 1
to belong to multiple groups, the first shall be added directly before the `<newline>` 1
character in the format string:

`" groups=%u(%s)"`

with the following arguments added at the end of the argument list:

<supplementary group ID>, *<supplementary group name>*

and the necessary number of the following added after that for any remaining supplementary group IDs:

```
",%u(%s)"
```

and the necessary number of the following arguments added at the end of the argument list:

<supplementary group ID>, *<supplementary group name>*

If any of the user ID, group ID, effective user ID, effective group ID, or 1
supplementary/multiple group IDs cannot be mapped by the system into printable 1
user or group names, the corresponding (`%s`) and name argument shall be omitted from the corresponding format string.

When any of the options are specified, the output format shall be as described under 4.30.3.

4.30.6.2 Standard Error

Used only for diagnostic messages.

4.30.6.3 Output Files

None.

4.30.7 Extended Description

None.

4.30.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred.

4.30.9 Consequences of Errors

Default.

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

Examples, Usage

The functionality provided by the 4BSD `groups` utility can be simulated using:

`id -Gn [`*user*`]`

Note that output produced by the `-G` option and by the default case could potentially produce very long lines on systems that support large numbers of supplementary groups. (On systems with user and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per name, 93 supplementary groups plus distinct effective and real group and user IDs could theoretically overflow the 2048-byte {LINE_MAX} text file line limit on the default output case. It would take about 186 supplementary groups to overflow the 2048-byte barrier using `id −G`.) This is not expected to be a problem in practice, but in cases where it is a concern, applications should consider using `fold −s` (see 4.25) before postprocessing the output of `id`.

History of Decisions Made

The 4BSD command `groups` was considered, but was not used as it did not provide the functionality of the `id` utility of the *SVID*. Also, it was thought that it would be easier to modify `id` to provide the additional functionality necessary to systems with multiple groups than to invent another command.

The options `−u`, `−g`, `−n`, and `−r` were added to ease the use of `id` with shell commands substitution. Without these options it is necessary to use some preprocessor such as `sed` to select the desired piece of information. Since output such as that produced by `id -u -n` is wanted frequently, it seemed desirable to add the options.

4.31 `join` — Relational database operator

4.31.1 Synopsis

`join` [`−a` *file_number* | `−v` *file_number*] [`−e` *string*] [`−o` *list*] [`−t` *char*]
  [`−1` *field*] [`−2` *field*] *file1 file2*

Obsolescent version:

`join` [`−a` *file_number*] [`−e` *string*] [`−j` *field*] [`−j1` *field*] [`−j2` *field*]
  [`−o` *list* . . .] [`−t` *char*] *file1 file2*

4.31.2 Description

The `join` utility shall perform an ‘‘equality join’’ on the files *file1* and *file2*. The joined files shall be written to the standard output.

The ‘‘join field’’ is a field in each file on which the files are compared. There shall be one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line by default shall consist of the join field, then the remaining fields from *file1*, then the remaining fields from *file2*. This format can be changed by using the `−o` option (see below). The `−a` option can be used to add unmatched lines to the output. The `−v` option can be used to output only unmatched lines.

By default, the files *file1* and *file2* should be ordered in the collating sequence of `sort −b` (see 4.58) on the fields on which they are to be joined, by default the first in each line. All selected output shall be written in the same collating sequence.

The default input field separators shall be `<blank>`s. In this case, multiple separators shall count as one field separator, and leading separators shall be ignored. The default output field separator shall be a `<space>`.

The field separator and collating sequence can be changed by using the `−t` option (see below).

If the input files are not in the appropriate collating sequence, the results are unspecified.

4.31.3 Options

The `join` utility shall conform to the utility argument syntax guidelines described in 2.10.2. The obsolescent version does not follow the utility argument syntax guidelines: the `−j1` and `−j2` options are multicharacter options and the `−o` option takes multiple arguments.

The following options shall be supported by the implementation:

`−a` *file_number*
: Produce a line for each unpairable line in file *file_number*, where *file_number* is `1` or `2`, in addition to the default output. If both `−a 1` and `−a 2` are specified, all unpairable lines shall be output.

`−e` *string*
: Replace empty output fields by string *string*.

`−j` *field*
: (Obsolescent.) Equivalent to: `−1` *field* `−2` *field*

`−j1` *field*
: (Obsolescent.) Equivalent to: `−1` *field*

`−j2` *field*
: (Obsolescent.) Equivalent to: `−2` *field*

`−o` *list*
: Construct the output line to comprise the fields specified in *list*, each element of which has the form *file_number*`.`*field*, where *file_number* is a file number and *field* is a decimal integer field number. The elements of *list* are either comma- or `<blank>`-separated, as specified in Guideline 8 in 2.10.2. The fields specified by *list* shall be written for all selected output lines. Fields selected by *list* that do not appear in the input shall be treated as empty output fields. (See the `−e` option.) The join field shall not be written unless specifically requested. The *list* shall be a single command line argument. However, as an obsolescent feature, the argument *list* can be multiple arguments on the command line. If this is the case, and if the `−o` option is the last option before *file1*, and if *file1* is of the form *string*.*string*, the results are undefined.

`−t` *char* Use character *char* as a separator, for both input and output. Every appearance of *char* in a line shall be significant. When this option is specified, the collating sequence should be the same as `sort` without the `−b` option.

`−v` *file_number*
Instead of the default output, produce a line only for each unpairable line in *file_number*, where *file_number* is 1 or 2. If both `−v 1` and `−v 2` are specified, all unpairable lines shall be output.

`−1` *field* Join on the *field* th field of file 1. Fields are decimal integers starting with 1.

`−2` *field* Join on the *field* th field of file 2. Fields are decimal integers starting with 1.

4.31.4 Operands

The following operands shall be supported by the implementation:

file1
file2 A pathname of a file to be joined. If either of the *file1* or *file2* operands is −, the standard input is used in its place.

4.31.5 External Influences

4.31.5.1 Standard Input

The standard input shall be used only if the *file1* or *file2* operand is −. See Input Files.

4.31.5.2 Input Files

The input files shall be text files.

4.31.5.3 Environment Variables

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

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 collating sequence `join` expects to have been used when the input files were sorted.

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

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

4.31.5.4 Asynchronous Events

Default.

4.31.6 External Effects

4.31.6.1 Standard Output

The `join` utility output shall be a concatenation of selected character fields. When the `−o` option is not specified, the output shall be:

`"%s%s%s\n"`, *<join field>* , *<other file1 fields>* , *<other file2 fields>*

If the join field is not the first field in either file, the *<other file fields>* are:

<fields preceding join field> , *<fields following join field>*

When the `−o` option is specified, the output format shall be:

`"%s\n"`, *<concatenation of fields>*

where the concatenation of fields is described by the `−o` option, above.

For either format, each field (except the last) shall be written with its trailing separator character. If the separator is the default (`<blank>`s), a single `<space>` character shall be written after each field (except the last).

4.31.6.2 Standard Error

Used only for diagnostic messages.

4.31.6.3 Output Files

None.

4.31.7 Extended Description

None.

4.31.8 Exit Status

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

0	All input files were output successfully.
>0	An error occurred.

4.31.9 Consequences of Errors

Default.

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

Examples, Usage

Pathnames consisting of numeric digits should not be specified directly following the `-o` list.

The developers of the standard believed that `join` should operate as documented in the *SVID* and BSD, not as historically implemented. Historical implementations do not behave as documented in these areas:

(1) Most implementations of `join` require using the `-o` option when using the `-e` option.

(2) Most implementations do not parse the `-o` option as documented, and parse the elements as separate *argv* items, until the item is not of the form *file_number* .*field* . This behavior is permitted as an obsolescent usage of the utility. To ensure maximum portability, *file1* should not be of the form *string* .*string* . A suitable alternative to guarantee portability would be to put the `--` flag before any *file1* operand.

The obsolescent `-j`, `-j1`, and `-j2` options have been described to show how they have been used in historical implementations. Earlier drafts showed `-j` *file_number field* , but a space was never allowed before the *file_number* and two option arguments were never intended.

History of Decisions Made

The ability to specify *file2* as `-` is not historical practice; it was added for completeness.

As a result of a balloting comment, the `-v` option was added to the nonobsolescent version. This option was felt necessary because it permitted the writing of *only* those lines that do not match on the join field, as opposed to the `-a` option, which prints both lines that do and do not match. This additional facility is parallel with the `−v` option of `grep`.

4.32 `kill` — Terminate or signal processes

4.32.1 Synopsis

`kill −s` *signal_name pid* . . .

`kill −l [`*exit_status* `]`

Obsolescent Versions :

`kill [`−*signal_name* `]` *pid* . . .

`kill [`−*signal_number* `]` *pid* . . .

4.32.2 Description

The `kill` utility shall send a signal to the process(es) specified by each *pid* operand.

For each *pid* operand, the `kill` utility shall perform actions equivalent to the POSIX.1 {8} *kill* () function called with the following arguments:

(1) The value of the *pid* operand shall be used as the *pid* argument.

(2) The *sig* argument is the value specified by the `−s` option, −*signal_number* option, or the −*signal_name* option, or by SIGTERM, if none of these options is specified.

4.32.3 Options

The `kill` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that in the obsolescent form, the −*signal_number* and −*signal_name* options are usually more than a single character.

The following options shall be supported by the implementation:

`−l` (The letter ell.) Write all values of *signal_name* supported by the implementation, if no operand is given. If an *exit_status* operand is given and it is a value of the `?` shell special parameter (see 3.5.2 and `wait` in 4.70) corresponding to a process that was terminated by a signal, the *signal_name* corresponding to the signal that terminated the process shall be written. If an *exit_status* operand is given and it is the unsigned decimal integer value of a signal number, the *signal_name* (the POSIX.1 {8}-defined symbolic constant name without the `SIG` prefix) corresponding to that signal shall be written. Otherwise, the results are unspecified.

`-s` *signal_name*
Specify the signal to send, using one of the symbolic names defined for Required Signals or Job Control Signals in POSIX.1 {8} 3.3.1.1. Values of *signal_name* shall be recognized in a case-independent fashion, without the `SIG` prefix. In addition, the symbolic name `0` shall be recognized, representing the signal value zero. The corresponding signal shall be sent instead of SIGTERM.

-*signal_name*
(Obsolescent.) Equivalent to `-s` *signal_name.*

-*signal_number*
(Obsolescent.) Specify a nonnegative decimal integer, *signal_number,* representing the signal to be used instead of SIGTERM, as the *sig* argument in the effective call to *kill* (). The correspondence between integer values and the *sig* value used is shown in the following table.

signal_number	*sig* Value
0	0
1	SIGHUP
2	SIGINT
3	SIGQUIT
6	SIGABRT
9	SIGKILL
14	SIGALRM
15	SIGTERM

The effects of specifying any *signal_number* other than those listed in the table are undefined.

In the obsolescent versions, if the first argument is a negative integer, it shall be interpreted as a -*signal_number* option, not as a negative *pid* operand specifying a process group.

4.32.4 Operands

The following operands shall be supported by the implementation:

pid A decimal integer specifying a process or process group to be signaled. The process(es) selected by positive, negative, and zero values of the *pid* operand shall be as described for POSIX.1 {8} *kill* () function. If the first *pid* operand is negative, it should be preceded by `--` to keep it from being interpreted as an option.

exit_status A decimal integer specifying a signal number or the exit status of a process terminated by a signal.

4.32.5 External Influences

4.32.5.1 Standard Input

None.

4.32.5.2 Input Files

None.

4.32.5.3 Environment Variables

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

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

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

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

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

4.32.5.4 Asynchronous Events

Default.

4.32.6 External Effects

4.32.6.1 Standard Output

When the `−l` option is not specified, the standard output shall not be used.

When the `−l` option is specified, the symbolic name of each signal shall be written in the following format:

`"%s%c"`, *<signal_name>* , *<separator>*

where the *<signal_name>* is in uppercase, without the `SIG` prefix, and the

<separator> shall be either a `<newline>` or a `<space>`. For the last signal written, *<separator>* shall be a `<newline>`.

When both the `–l` option and *exit_status* operand are specified, the symbolic name of the corresponding signal shall be written in the following format:

`"%s\n"`, *<signal_name>*

4.32.6.2 Standard Error

Used only for diagnostic messages.

4.32.6.3 Output Files

None.

4.32.7 Extended Description

None.

4.32.8 Exit Status

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

- 0 At least one matching process was found for each *pid* operand, and the specified signal was successfully processed for at least one matching process.
- >0 An error occurred.

4.32.9 Consequences of Errors

Default.

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

Examples, Usage

Any of the commands

```
kill –9 100 -165
kill –s kill 100 -165
kill –s KILL 100 -165
```

sends the SIGKILL signal to the process whose process ID is `100` and to all processes whose process group ID is `165`, assuming the sending process has permission to send that signal to the specified processes, and that they exist.

POSIX.1 {8} and POSIX.2 do not require specific signal numbers for any *signal_names*. Even the *–signal_number* option provides symbolic (although numeric) names for signals. If a process is terminated by a signal, its exit status indicates the signal that killed it, but the exact values are not specified. The `kill −l` option, however, can be used to map decimal signal numbers and exit status values into the name of a signal. The following example reports the status of a terminated job:

```
job
stat=$?
if [ $stat -eq 0 ]
then
        echo job completed successfully.
elif [ $stat -gt 128 ]
then
        echo job terminated by signal SIG$(kill -l $stat).
else
        echo job terminated with error code $stat.
fi
```

History of Decisions Made

The signal name extension was based on a desire to avoid limiting the `kill` utility to implementation-dependent values.

The `−l` option originated from the C-shell, and is also implemented in the KornShell. The C-shell output can consist of multiple output lines, because the signal names do not always fit on a single line on some terminal screens. The KornShell output also included the implementation-specific signal numbers, and was felt by the working group to be too difficult for scripts to parse conveniently. The specified output format is intended not only to accommodate the historical C-shell output, but also to permit an entirely vertical or entirely horizontal listing on systems for which this is appropriate.

An earlier draft invented the name SIGNULL as a *signal_name* for signal 0 (used by POSIX.1 {8} to test for the existence of a process without sending it a signal). Since the *signal_name* `"0"` can be used in this case unambiguously, SIGNULL has been removed.

An earlier draft also required symbolic *signal_name* s to be recognized with or without the `SIG` prefix. Historical versions of `kill` have not written the `SIG` prefix for the `−l` option and have not recognized the `SIG` prefix on *signal_name* s. Since neither application portability nor ease of use would be improved by requiring this extension, it is no longer required.

POSIX.2 contains no utility that browses for process IDs. Values for *pid* are available via the `!` and `$` parameters of the shell command language (see 3.5.2).

The use of numeric signal values was the subject of a long debate in the Working Group. During balloting, it was determined that their use should be declared obsolescent, but retained to provide backward compatibility to existing applications.

Existing implementations of `kill` permit negative *pid* operands representing process groups, but this was often unclearly documented. The assumption that an initial negative number argument specifies a signal number (rather than a process group) is the existing behavior, and was retained. Therefore, to send the default signal to a process group (say 123), an application should use a command similar to one of the following:

```
kill -TERM -123
kill -- -123
```

The `−s` option was added in response to international interest in providing some form of `kill` that meets the Utility Syntax Guidelines.

Some implementations provide `kill` only as a shell built-in utility and use that status to support the extension of killing background asynchronous lists (those started with &), by the use of job identifiers. For example,

```
kill %1
```

would `kill` the first asynchronous list in the background. This standard does not require (but permits) such an extension, because other related job-control features are not provided by the shell, and because these facilities are not ordinarily usable in portable shell applications. This notation is expected to be introduced by the UPE.

4.33 `ln` — Link files

4.33.1 Synopsis

`ln [−f]` *source_file target_file*

`ln [−f]` *source_file* . . . *target_dir*

4.33.2 Description

In the first synopsis form, the `ln` utility shall create a new directory entry (link) for the file specified by the *source_file* operand, at the *destination* path specified by the *target_file* operand. This first synopsis form shall be assumed when the final operand does not name an existing directory; if more than two operands are specified and the final is not an existing directory, an error shall result.

In the second synopsis form, the `ln` utility shall create a new directory entry for each file specified by a *source_file* operand, at a *destination* path in the existing directory named by *target_dir*.

If the last operand specifies an existing file of a type not specified by POSIX.1 {8}, the behavior is implementation defined.

The corresponding destination path for each *source_file* shall be the concatenation of the target directory pathname, a slash character, and the last pathname component of the *source_file*. The second synopsis form shall be assumed when the final operand names an existing directory.

For each *source_file* :

(1) If the *destination* path exists:

(a) If the `-f` option is not specified, `ln` shall write a diagnostic message to standard error, do nothing more with the current *source_file,* and go on to any remaining *source_files.*

(b) Actions shall be performed equivalent to the POSIX.1 {8} *unlink* () function, called using *destination* as the *path* argument. If this fails for any reason, `ln` shall write a diagnostic message to standard error, do nothing more with the current *source_file,* and go on to any remaining *source_files.*

(2) Actions shall be performed equivalent to the POSIX.1 {8} *link* () function using *source_file* as the *path1* argument, and the *destination* path as the *path2* argument.

4.33.3 Options

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

The following option shall be supported by the implementation:

`-f` Force existing *destination* pathnames to be removed to allow the link.

4.33.4 Operands

The following operands shall be supported by the implementation:

source_file A pathname of a file to be linked. This can be a regular or special file; whether a directory can be linked is implementation defined.

target_file The pathname of the new directory entry to be created.

target_dir A pathname of an existing directory in which the new directory entries are to be created.

4.33.5 External Influences

4.33.5.1 Standard Input

None.

4.33.5.2 Input Files

None.

4.33.5.3 Environment Variables

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

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

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

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

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

4.33.5.4 Asynchronous Events

Default.

4.33.6 External Effects

4.33.6.1 Standard Output

None.

4.33.6.2 Standard Error

Used only for diagnostic messages.

4.33.6.3 Output Files

None.

4.33.7 Extended Description

None.

4.33.8 Exit Status

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

0 All the specified files were linked successfully.

>0 An error occurred.

4.33.9 Consequences of Errors

Default.

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

Examples, Usage

None.

History of Decisions Made

Some historic versions of `ln` (including the one specified by the *SVID*) unlink the destination file, if it exists, by default. If the mode does not permit writing, these versions will prompt for confirmation before attempting the unlink. In these versions the `-f` option causes `ln` to not attempt to prompt for confirmation.

This allows `ln` to succeed in creating links when the target file already exists, even if the file itself is not writable (although the directory must be). Previous versions of this draft specified this functionality.

This draft does not allow the `ln` utility to unlink existing destination paths by default for the following reasons:

— The `ln` utility has traditionally been used to provide locking for shell applications, a usage that is incompatible with `ln` unlinking the destination path by default. There was no corresponding technical advantage to adding this functionality.

— This functionality gave `ln` the ability to destroy the link structure of files, which changes the historical behavior of `ln`.

— This functionality is easily replicated with a combination of `rm` and `ln`.

— It is not historical practice in many systems; BSD and BSD-derived systems do not support this behavior. Unfortunately, whichever behavior is selected can cause scripts written expecting the other behavior to fail.

— It is preferable that `ln` perform in the same manner as the *link*() function, which does not permit the target to already exist.

This standard retains the `−f` option to provide support for shell scripts depending on the *SVID* semantics. It seems likely that shell scripts would not be written to handle prompting by `ln`, and would therefore have specified the `−f` option.

It should also be noted that `−f` is an undocumented feature of many historical versions of the `ln` utility, allowing linking to directories. These versions will require modification.

Previous drafts of this standard also required an `−i` option, which behaved like the `−i` options in `cp` and `mv`, prompting for confirmation before unlinking existing files. This was not historical practice for the `ln` utility and has been deleted from this version.

Although symbolic links are not part of the standard, the `−s` option should be used only for the traditional purpose of creating symbolic links.

4.34 `locale` — Get locale-specific information

4.34.1 Synopsis

`locale [ −a | −m ]`

`locale [−ck]` *name* ...

4.34.2 Description

The `locale` utility shall write information about the current locale environment, or all public locales, to the standard output. For the purposes of this clause, a *public locale* is one provided by the implementation that is accessible to the application.

When `locale` is invoked without any arguments, it shall summarize the current locale environment for each locale category as determined by the settings of the environment variables defined in 2.5.

When invoked with operands, it shall write values that have been assigned to the keywords in the locale categories, as follows:

— Specifying a keyword name shall select the named keyword and the category containing that keyword.

— Specifying a category name shall select the named category and all keywords in that category.