-- --

IEEE P1003.2b Draft 2 − December 1991

Information technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 2

Section 1: Revisions to General

1.1 Scope

HLJ : To be provided. See the Introduction. Proposed text is solicited.

1.2 Normative References

⇒ **1.2 Normative References.** *Add the following entries to the Normative References clause in the correct sorted sequence, disregarding the sequence numbers shown here.*

{9} ISO 1001: 1986, *Information processing—File structure and labelling of magnetic tapes for information interchange.*

Issue: *The following three standards seem to correspond to ANSI X3.14, 22, 39, and 54. Someone with access to the actual documents should check this assumption carefully.*

{10} ISO 1863: 1976, *Information processing—9-track, 12,7 mm (0.5 in) wide magnetic tape for information interchange recorded at 32 rpmm (800 rpi).*

{11} ISO 3788: 1976, *Information processing—9-track, 12,7 mm (0.5 in) wide magnetic tape for information interchange recorded at 63 rpmm (1600 rpi).*

{12} ISO 5652: 1984, *Information processing—9-track, 12,7 mm (0.5 in) wide magnetic tape for information interchange—Format and recording, using group coding at 246 cpmm (6250 cpi).*

{13} ISO/IEC 10646: . . . ,[1] *Information technology—Universal Coded Character Set (UCS).*

1.3 Conformance

There are no revisions to this clause (yet?).

HLJ : There has been some discussion within the profile groups about breaking up the full Section 4 into smaller groups of required utilities. This is the place to do that.

1.4 Test Methods 2

To Be Provided. 2

1) To be approved and published.

Section 2: Revisions to Terminology and General Requirements

Editor's Note: This section is new for Draft 3. It is not further diff-marked. 3

Editor's Note: The following material on symbolic links is related to P1003.1a/D5; the definition is from that draft verbatim. All of the symbolic link material in this and later sections is contingent on P1003.1a being approved before P1003.2b; if it is not, all of this may be put into an informative annex. When P1003.1a is approved, a number of the P1003.2 definitions copied from POSIX.1 {8} will be updated automatically.

2.2.2 General Terms

⇒ **2.2.2 General Terms.** *Modify the contents of subclause 2.2.2, General Terms, to add the following definitions in the correct sorted order [disregarding the subclause numbers shown here].*

2.2.2.201 symbolic link: A type of file that contains a pathname.

The pathname is interpolated into a pathname being resolved, during pathname resolution, to create a new pathname when it is encountered. [P1003.1a/D5]

2.9.2 Pathname Resolution

⇒ **2.9.1.8 Pathname Resolution.** *Add a new sentence at the end of the subclause:*

When resolving the pathname of file being created, the effects of symbolic links shall be undefined unless the {POSIX2_SYMLINKS} variable is in effect for the directory in which the link would be created.

Editor's Note: The following rationale will be added to E.2.9.1.8, but is kept here for this draft:

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

P1003.1a now includes symbolic links in pathname resolution and a number of concepts are automatically inherited in POSIX.2 by this inclusion. The large majority of standard utilities resolve pathnames and operate on files without special arrangements for symbolic links. Because of the global POSIX.1 {8} inheritance, this entails very few modifications to utility descriptions.

2.11.3 Operands

⇒ **2.11.4 Operands (POSIX.1: Def)** *Add a new paragraph at the end of the subclause:*

> All of the standard utilities shall be consistent in their processing of symbolic links specified directly on the command line versus those found elsewhere in the file hierarchy as a consequence of their operation.

Editor's Note: The following rationale will be added to E.2.11.4, but is kept here for this draft:

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

In most cases, the standard utilities follow symbolic links to access individual files, but when they traverse file hierarchies, they stop descending when they reach a symbolic link. (The developers of the standard use the informal term ''physical walk'' to describe this case.) There are stated exceptions to this (''logical walks''), such as `cp` and `du`. However, none of the utilities treat their command-line arguments any differently than their encountered files. For example, `find` does not follow symbolic links by default. In the following example:

```
find /foo -print
```

if `/foo` is a symbolic link, no output will result, because `find` treats its arguments consistently with files found during its hierarchy traversal. When modified to:

```
find /foo -follow -print
```

the output would include `/foo` from the command line and all of the files below it, including those found in traversing symbolic links. Not all historical practice is as consistent as the POSIX.2 standard utilities in this regard.

2.13.3 Pathname Variable Values

⇒ **2.13.3 Pathname Variable Values.** *Add a new subclause, 2.13.3, Pathname Variable Values, as follows:*

The values in Table 2-100 may be constants within an implementation or may vary from one pathname to another.

Table 2-100 – Pathname Variable Values

Name	Description
{POSIX2_SYMLINKS}	When referring to a directory. the system supports the creation of symbolic links within that directory; for nondirectory files, the meaning of {POSIX2_SYMLINKS} is undefined.

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

The {POSIX2_SYMLINKS} variable indicates that the underlying operating system supports the creation of symbolic links in specific directories. Many of the POSIX.2 utilities that deal with symbolic links do not depend on this value. For example, a utility that follows symbolic links (or does not, as the case may be) will only be affected by a symbolic link if it encounters one. Presumably, a file system that does not support symbolic links will not contain any. This variable does affect such utilities as `ln -s` and `pax` that attempt to create symbolic links.

{POSIX2_SYMLINKS} was developed even though there is no comparable configuration value in P1003.1a. Since POSIX.2 does not depend on a fully conforming POSIX.1 {8} system underneath, the developers of the standard wished to allow systems in which this was an optional feature, perhaps on a file system basis.

Section 3: Revisions to Shell Command Language

1 There are no revisions to Section 3 (yet).

Section 4: Revisions to Execution Environment Utilities

4.5 cd — Change working directory

Editor's Note: The following rationale will be added to E.4.5, but is kept here with `cd` *for this draft:*

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

Some historical shells, such as the KornShell, took special actions when the directory name contained a dot-dot component, selecting the logical parent of the directory, rather than the actual parent directory (i.e., it moved up one level toward the / in the pathname, remembering what the user typed, rather than performing the equivalent of the C-language call

```
chdir("..");
```

In such a shell, the following commands would not be equivalent for all directories:

```
cd .. && ls                ls ..
```

This behavior is not consistent with the definition of dot-dot and is not allowed by this standard, although it would be allowed as an extension if triggered by an environment variable or command-line option.

4.6 `chgrp` — Change file group ownership

⇒ **4.6.3 `chgrp` Options.** *Add the following sentence to the end of the paragraph describing the* `−R` *option:*

> When a symbolic link is encountered in the file hierarchy, `chgrp` shall modify the file referenced by the link and shall continue with the other files in the directory containing the link, not attempting to follow the link to another portion of the file hierarchy.

Editor's Note: The following rationale will be added to E.4.6, but is kept here with `chgrp` *for this draft:*

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

When `chgrp` is given a symbolic link as a *file* operand, it changes the file resolved from the link. This is a consequence of the dependence on the *chown* () functionality, per the Description. The POSIX.1 {8} *chown* () function does not modify the link itself [and provides no *lchown* () function that does that]. However, in traversing directories for the `−R` option, it does not follow links into other parts of the hierarchy. This is based on historical practice. Some systems provide a `−h` option that does modify the link itself and this is a valid extension, but it has been omitted from POSIX.2 because POSIX.1 {8} does not support it. No known systems cause `−R` to follow symbolic links; this functionality is possible with

```
find ... -follow ... | xargs chgrp ...
```

4.7 `chmod` — Change file modes

⇒ **4.7.3 `chmod` Options.** *Add the following sentence to the end of the paragraph describing the* `–R` *option:*

When a symbolic link is encountered in the file hierarchy, `chmod` shall modify the file referenced by the link and shall continue with the other files in the directory containing the link, not attempting to follow the link to another portion of the file hierarchy.

⇒ **4.7.4 `chmod` Operands.** *Add the following sentence to the end of the paragraph describing the file operand:*

If *file* is a symbolic link, `chmod` shall change the mode of the file referenced by the symbolic link, not the symbolic link itself.

Editor's Note: The following rationale will be added to E.4.7, but is kept here with `chmod` *for this draft:*

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

When `chmod` is given a symbolic link as a *file* operand, it changes the file resolved from the link. Unlike the `chgrp` and `chown` descriptions, where this behavior is a consequence of the dependence on the *chown* () functionality, this must be explicitly stated for `chmod`. The POSIX.1 {8} *chmod* () function, on which many `chmod` implementations will be based, does not modify the link itself [and provides no *lchmod* () function that does that]. However, in traversing directories for the `–R` option, it does not follow links into other parts of the hierarchy. No known systems cause `–R` to follow symbolic links; this functionality is possible with

```
find ... -follow ... | xargs chmod ...
```

4.8 `chown` — Change file ownership

⇒ **4.8.3 `chown` Options.** *Add the following sentence to the end of the paragraph describing the* `−R` *option:*

When a symbolic link is encountered in the file hierarchy, `chown` shall modify the file referenced by the link and shall continue with the other files in the directory containing the link, not attempting to follow the link to another portion of the file hierarchy.

Editor's Note: The following rationale will be added to E.4.8, but is kept here with `chown` *for this draft:*

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

When `chown` is given a symbolic link as a *file* operand, it changes the file resolved from the link. This is a consequence of the dependence on the *chown* () functionality, per the Description. The POSIX.1 {8} *chown* () function does not modify the link itself [and provides no *lchown* () function that does that]. However, in traversing directories for the `−R` option, it does not follow links into other parts of the hierarchy. This is based on historical practice. Some systems provide a `−h` option that does modify the link itself and this is a valid extension, but it has been omitted from POSIX.2 because POSIX.1 {8} does not support it. No known systems cause `−R` to follow symbolic links; this functionality is possible with

```
find ... -follow ... | xargs chown ...
```

4.13 `cp` — Copy files

⇒ **4.13.2 `cp` Description.** *Add the following sentence to the end of the first sentence of the seventh paragraph (beginning* ‘‘In the following description, source_file refers . . . ’’*):*

When *source_file* is a symbolic link, `cp` shall take actions based on the file or directory referenced by the link, not the contents of the link itself. In traversing the file hierachy with `−R` or `−r`, it also shall follow symbolic links to other parts of the hierarchy.

Editor's Note: Question for working group: Should the language about infinite loop detection be copied from `find` *and* `pax`*?*

Editor's Note: The following rationale will be added to E.4.13, but is kept here with `cp` *for this draft:*

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

When `cp` encounters a symbolic link to be copied, it copies the contents of the file resolved from the link. Unlike many of the other utilities, such as `chown` or the default case of `find`, `cp −r` and `−R` do traverse directories by following links into other parts of the hierarchy. This allows the data in a file hierarchy to be duplicated at another place, such as for backup purposes.

4.24 `find` — Find files

⇒ **4.24.4 `find` Operands.** *Add the following primary in the proper sorted order:*

`−follow` The primary always shall evaluate as true; it shall cause `find` to traverse directories by following symbolic links into other parts of the hierarchy. By default, `find` shall not follow symbolic links. In following symbolic links, `find` shall keep track of the directories visited so that it can detect infinite loops. (For example, a loop would occur if a symbolic link refers to a directory that is an ancestor.) When it detects a loop, `find` shall write a diagnostic message to standard error and shall either recover its position in the hierarchy in an implementation-defined manner or terminate.

⇒ **4.24.4 `find` Operands.** *In the* `−type` *c description, add the character* `l` *(ell) to represent a symbolic link.*

4.33 `ln` — Link files

⇒ **4.33.1 `ln` Synopsis.** *Modify the Synopsis to be:*

`ln [−f]` *source_file target_file*

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

`ln −s` *source_file target_file*

`ln −s` *source_file ... target_dir*

⇒ **4.33.2 `ln` Description.** *Modify the entire Description subclause as follows:*

The *ln* () utility creates either hard or symbolic links, depending on the presence of the `−s` option: hard links shall be created by default, symbolic links with `−s`.

4.33.2.1 Creating Hard Links

<current contents of 4.33.2 ...>

If *source_file* is a symbolic link, `ln` shall follow the link and create a hard link at *destination* to the file referenced by *source_file* . If either *target_file* or *target_dir* is a symbolic link, the third or fourth synopsis forms (respectively) shall be assumed, even though `−s` was not specified; use of `−f` in these cases is undefined.

4.33.2.2 Creating Symbolic Links

In the third synopsis form, the `ln` utility shall create a symbolic link for the file specified by the *source_file* operand, at the *destination* path specified by the *target_file* operand. If *target_file* is a symbolic link to a nondirectory file, `ln` shall overwrite the symbolic link. This third synopsis form shall be assumed when there are two operands and the second is not an existing directory.

In the fourth synopsis form, the `ln` utility shall create a symbolic link for each file specified by a *source_file* operand, at a *destination* path in the existing directory named by *target_dir*. 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.* If *target_dir* is a symbolic link to a directory, `ln` shall follow the link and create a symbolic link to *source_file* in the directory *target_dir*.

For both the third and fourth synopsis forms:

— If the directory in which the destination is to reside does not support {POSIX 2_SYMLINKS }, results are undefined.

— The *source_file* operand can be any pathname and need not represent an existing or accessible file.

— The *source_file* and *destination* need not reside on the same file system.

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

For each *source_file*:

(1) If the *destination* path exists, `ln` shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file* s.

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

If *source_file* is itself a symbolic link, `ln` shall not follow the link, but shall create a symbolic link in *destination* to *source_file* (the symbolic link) itself.

⇒ **4.33.3 `ln` Options.** *Add the following option in the proper sorted order:*

`-s` Create symbolic links instead of the default hard links.

⇒ **4.33.4 `ln` Operands.** *Replace all of the Operands subclause with:*

The following operands shall be supported by the implementation:

source_file A pathname of a file to be linked. When `-s` is not specified this can be a regular or special file; whether a directory can be linked is implementation defined. When `-s` is specified, any pathname can be used.

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

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

HLJ: Is it correct that `-f` *cannot be used with* `-s`*?*

4.39 `ls` — List directory contents

⇒ **4.39.1 `ls` Synopsis.** *Modify the Synopsis to be:*

`ls [−CFLRacdilqrtu1][`*file* ...`]`

⇒ **4.39.3 `ls` Options.** *Replace the description of the* `−F` *option with the following:*

`−F` Write a slash (`/`) immediately after each pathname that is a directory, an asterisk (∗) after each that is executable, a vertical bar (`|`) after each that is a FIFO, and an at-sign (@) after each that is a symbolic link.

⇒ **4.39.3 `ls` Options.** *Add the following option in the proper sorted order:*

`−L` When a symbolic link is encountered, write the name of the file or directory the link refers to, rather than the name of the link itself.

⇒ **4.39.6.1 `ls` Standard Output.** *Replace the six-line description of* `−l` *(beginning with* ''If the −l option is specified, ...'')*with:*

If the `−l` option is specified without `−L`, the following information shall be written:

`"%s %u %s %s %u %s %s\n"`, *<file mode>*, *<number of links>*, *<owner name>*, *<group name>*, *<number of bytes in the file>*, *<date and time>*, *<pathname>*

If the file is a symbolic link, this information shall be about the link itself and the *<pathname>* field shall be of the form:

`"%s -> %s"`, *<name of link>*, *<contents of link>*

If both `−l` and `−L` are specified, the following information shall be written:

`"%s %u %s %s %u %s %s\n"`, *<file mode>*, *<number of links>*, *<owner name>*, *<group name>*, *<number of bytes in the file>*, *<date and time>*, *<pathname of link>*

where all fields except *<pathname of link>* shall be for the file resolved from the symbolic link.

In both of the preceding `−l` forms, if *<owner name>* or *<group name>* cannot be determined, they shall be replaced with their associated numeric values using the format `"%u"`.

⇒ **4.39.6.1 `ls` Standard Output.** *Add the following to the list of <entry type> characters:*

`l` (ell) Symbolic link

4.43 `mv` — Move files

⇒ **4.43.2 `mv` Description.** *Replace the first two paragraphs of the Description with:*

In the first synopsis form, the `mv` utility shall move the file named by the *source_file* operand to the *destination* specified by the *target_file*. This first synopsis form is assumed when the final operand does not name an existing directory and is not a symbolic link referring to an existing directory.

In the second synopsis form, `mv` shall move each file named by a *source_file* operand to a *destination* file in the existing directory named by the *target_dir* operand, or referenced if *target_dir* is a symbolic link. The *destination* path for each *source_file* shall be the concatenation of the target directory, a single slash character, and the last pathname component of the *source_file*. This second form is assumed when the final operand names an existing directory.

⇒ **4.43.2 `mv` Description.** *Replace the first sentence of item (5) with:*

The file hierarchy rooted in *source_file* shall be duplicated as a file hierarchy rooted in the destination path. If *source_file* or any of the files below it in the hierarchy are symbolic links, the links themselves shall be moved, rather than any files they refer to.

Editor's Note: The following rationale will be added to E.4.43, but is kept here with `mv` for this draft:

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

When `mv` is dealing with a single file system and *source_file* is a symbolic link, the link itself is moved as a consequence of the dependence on the POSIX.1 {8} *rename*() functionality, per the Description. Across file systems, this has to be made explicit.

4.48 **pax** — Portable archive interchange

⇒ **4.48.3 pax Options.** *Add the following option in the proper sorted order:* 3

`−L` Follow all symbolic links by traversing the the pathnames that the 3
links reference. In following symbolic links, `pax` shall keep track of 3
the directories visited so that it can detect infinite loops. (For exam- 3
ple, a loop would occur if a symbolic link refers to a directory that is 3
an ancestor.) When it detects a loop, `pax` shall write a diagnostic 3
message to standard error and shall either recover its position in the 3
hierarchy in an implementation-defined manner or terminate. The 3
default behavior shall be to archive the symbolic link itself. 3

⇒ **4.48.3 pax Options.** *Modify the description of the* `−o` *options option as follows:*

`−o` *options*

Provide information to the implementation to modify the algorithm for extracting or writing files that is specific to the file format specified by `−x`. The value of *options* shall consist of one or more comma-separated keywords of the form:

keyword `[=`*value* `][,`*keyword* `[=`*value* `], ...]`

Within the optional *value* field, the application shall precede any literal comma with a backslash, which shall be ignored, but preserves the comma as part of *value* . Multiple `−o` options can be specified. The following keyword values of *options* shall be supported for the file formats as indicated:

`charmap=`*name*

HLJ : I have included a modification of the POSIX. 2 Draft 11 `−e` *charmap option to get the working group started on the pathname translation problem. I am not personally recommending this method, but WG15 has indicated that something similar would be necessary.*

The *name* value shall represent either a system-provided charmap description (see 2.4.1) or the pathname of an application-provided charmap file.

With the `−w` option, filenames shall be written using only characters in the portable character set (except NUL ; see Table 2-3). Any character in the filename that is not in the portable character set shall be replaced by `pax` using the following steps; the description refers to the escape character defined in the named *charmap* (see the description of `escape_char` in 2.4.1):

(1) Any less-than, greater-than, and escape characters shall have an escape character inserted into the filename immediately before the character.

(2) Any character in the filename that is not part of the portable character set shall be replaced with a string consisting of a less-than character, the symbolic name of the character from the charmap, and a greater-than character.

(3) Any escape, less-than, or greater-than characters in the replacing symbolic names shall be modified as in step (1).

If a character for which no corresponding symbolic name exists in the charmap is found in a filename, `pax` shall write a diagnostic message to standard error; whether the file is written to the archive is unspecified.

With the `−r` option, or if neither the `−r` or `−w` option is specified, the following steps shall be used:

(1) Any sequences of a less-than character, not immediately preceded by an odd number of escapes, a string, and a greater-than character not immediately preceded by an odd number of escapes, shall be modified 2
as follows: 2

(a) The leading less-than character and trailing greater-than character shall be removed.

(b) Any sequences of an escape followed by an escape, less-than, or greater-than character shall have the leading escape character removed.

(c) The sequence shall be replaced with the encoded value of the character represented by the resulting symbolic name, from the charmap. If no encoding exists, the modifications to the sequence are implementation defined.

(2) Any sequences of an escape followed by an escape, less-than, or greater-than character shall have the leading escape character removed.

`linkdata`

(Applicable to the `−x` *pax* format.) When the `−w` option is given, but the `−r` is not, write the contents of a file to the archive even when that file is merely a hard link to a file whose contents have already been written to the archive.