IEEE P1003.2a Draft 8 – December 1991

Information technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: User Portability Extension (UPE)

Section 1: Revisions to General

⇒ **1.1 Scope.** *Modify the contents of clause 1.1, Scope, as shown below.*

1.1 Scope

This standard defines a standard source code level interface to command interpretation, or ''shell,'' services and common utility programs for application programs. These services and programs are complementary to those specified by ISO/IEC 9945-1: 1990 {8}, hereinafter referred to as 'POSIX.1 {8}.'' When the User Portability Utilities Option is included, this standard has additional scope. The list of utilities and features is extended to provide a common interactive environment for system users and program developers.

The standard has been designed to be used by both application programmers and system implementors. When the User Portability Utilities Option is included, it is also to be used by system users and program developers. However, it is intended to be a reference document and not a tutorial on the use of the services, the utilities, or the interrelationships between the utilities.

The emphasis of this standard without the User Portability Utilities Option is on the shell and utility functionality required by application programs (including ‘‘shell scripts’’) and not on the direct interactive use of the shell command language or the utilities by humans. When the User Portability Utilities Option is included, the emphasis is extended to support terminal users in a consistent manner across all conforming systems. There are three constraining factors that limit this user portability scope:

(1) The *users* in this context are limited to the group of individuals who are familiar with the style of interaction characteristic of historically derived systems based on one of the UNIX operating systems. Typical users would include program developers, engineers, or general purpose time-sharing users.

(2) The *environment* to be supported is a multi-user time-sharing system supporting character-oriented display terminals. Alternatively, it is a collection of single-user systems interconnected via local area networks or telephone lines, but with similar user interfaces. This standard does not include support tailored for bit-mapped or graphics display terminals, although it is expected that such terminals could emulate the character orientation required by this environment. When facilities require cursor addressability from the terminal hardware, these are specifically identified in the standard.

(3) The facilities to be provided are based on the historical models of the following documents: the *System V Interface Definition* {B24}; the BSD User Manual {B28}; the *X/Open Portability Guide* {B31}; and, documentation for the KornShell {B25}. Emphasis will be placed on standardizing existing practice for existing users, with changes or additions limited to correcting deficiencies in the following areas:

    (a) Support for international character sets and other localization requirements, such as date formats, collation sequences, etc.;

    (b) Reconciliation of differences between the historical implementations;

    (c) Elimination of system or device dependencies;

    (d) Corrections of features that could reduce system or user security/integrity.

Portions of this standard comprise optional language bindings to system service interfaces. See, for example, the C Language Bindings Option in Annex B. This standard is intended to describe language interfaces and utilities in sufficient detail so that an application developer can understand the required interfaces without access to the source code of existing implementations on which they may be based. Therefore, it does not attempt to describe the source programming language or internal design of the utilities; they should be considered ‘‘black boxes’’ that exhibit the described functionality.

For language interfaces, or functions, this standard has been defined exclusively at the source code level. The objective is that a conforming portable application source program can be translated to execute on a conforming implementation. The standard assumes that the source program may need to be retranslated to produce target code for a new environment prior to execution in that environment.

There is no requirement that the base operating system supporting the shell and utilities be one that fully conforms to ISO/IEC 9945-1: 1990 {8}. (The base system could contain a subset of POSIX.1 {8} functionality, enough to support the requirements for this standard, as described in 2.9.1, but that could not claim full conformance to all of POSIX.1 {8}.) Furthermore, there is no requirement that the shell command interpreter or any of the standard utilities be written as POSIX.1 {8} conforming programs, or be written in any particular language.

Although not requiring a fully conforming POSIX.1 {8} base, this standard is based upon documentation and the knowledge of existing programs that assume an interface and architecture similar to that described by POSIX.1 {8}. Any questions regarding the definition of terms or the semantics of an underlying concept should be referred to POSIX.1 {8}.

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

The purpose of this supplement is to extend the benefits of the common shell and tools environment from the application program to the interactive system user, by defining optional facilities in a supplement to the original standard, a ‘‘User Portability Extension.’’ The original standard’s scope limited the selection of utilities and features to those useful in a noninteractive environment only. Historically, however, many of the tools described in the original standard have seen interactive use; this supplement is predicated on the need to offer a completely consistent interactive environment for the users of such tools.

The scope was carefully worded to exclude the development of interfaces for new technology environments, such as bit-mapped graphics windowing terminals, and ‘‘look-and-feel’’ considerations (icons, menus, mice, etc.) for novice or specialized system users. There are numerous other standards groups in this domain and POSIX.2 did not want to overlap, but merely satisfy historical users.

The acknowledgment of four base documents placed the UPE effort squarely within guidelines established by the developers of the standard (the IEEE Technical Committee on Operating Systems and Operational Environments) to anchor proposed standards on industry practice and documentation. Since a primary purpose was to carry users familiar with historical practice along into a more portable open systems environment, the base documents were closely adhered to in terms of utility names, features, and behaviors. Only the four exception criteria called for any modifications to historical practice.

Selecting the KornShell documentation was a clear signal by the working group 8
that `ksh` extended scripting features and interactive command-line editing 8
features were major factors in selecting POSIX shell behavior. The System V shell 8
and the C-shell are also in the base documents, but numerous working group dis- 8
cussions resulted in the following guidelines used to prepare and ballot the drafts: 8

— Use the System V shell as the broad shell scripting base, but select KornShell features to correct historical deficiencies (such as the lack of easy arithmetic and substring manipulation, the tilde expansion, and the problems of command substitution nesting).

— Use the KornShell model for interactive features such as command-line editing.

— When required features appeared in both the C-shell and KornShell, emphasize the KornShell interface unless it was clearly deficient. This decision was predicated on the fact that it made less sense to select scripting features and interactive features from two very different shells. Historical C-shell users would find the POSIX shell to be rather different than their experience and selecting isolated C-shell features would only serve to confuse both C-shell and KornShell users.

1.3.1.3 Conforming Implementation Options

⇒ **1.3.1.3 Conforming Implementation Options.** *Modify the contents of subclause 1.3.1.3, to add the following table entries in the proper sorted order:*

{POSIX 2_CHAR _TERM }	The system supports at least one terminal type capable of all operations described in this standard. See 2.14.
{POSIX 2_UPE }	The system supports the User Portability Utilities Option in Section 5.

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

See the rationale for 2.14.

Section 2: Revisions to Terminology and General Requirements

2.2.1 Terminology

⇒ **2.2.1 Terminology.** *Modify the contents of subclause 2.2.1, Terminology, to add the following definition in the correct sorted order [disregarding the subclause number shown here].*

2.2.1.1 application: When the User Portability Utilities Option is supported, requirements associated with the term *application* shall also be interpreted to include the actions of the user who is interacting with the system by entering shell command language statements from a terminal.

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

The term *application* is not defined in either POSIX.1 {8} or POSIX.2; it is assumed to be a part of general computer science terminology. The definition in this subclause is present to ensure readers understand that the general requirements for applications also apply to interactive users. If, for instance, POSIX.2 requires a Strictly Conforming POSIX.2 Application to use a specific option to a utility, an interactive user would also have to use that option to be considered strictly conforming.

2.2.2 General Terms

⇒ **2.2.2 General Terms.** *Modify the following two subclauses of 2.2.2, General Terms, to add the synonym shown for each:*

2.2.2.10 background process group [background job]: Any process group, other than a foreground process group, that is a member of a session that has established a connection with a controlling terminal. [POSIX.1 {8}]

2.2.2.73 foreground process group [foreground job]: A process group whose member processes have certain privileges, denied to processes in background process groups, when accessing their controlling terminal.

Each session that has established a connection with a controlling terminal has exactly one process group of the session as the foreground process group of that controlling terminal. See POSIX.1 {8}7.1.1.4. [POSIX.1 {8}]

⇒ **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 alert: To cause the user's terminal to give some audible or visual indication that an error or some other event has occurred.

When the standard output is directed to a terminal device, the method for alerting the terminal user is unspecified. When the standard output is not directed to a terminal device, the alert shall be accomplished by writing the `<alert>` character to standard output (unless the utility description indicates that the use of standard output produces undefined results in this case).

2.2.2.202 block-mode terminal: A terminal device operating in a mode incapable of the character-at-a-time input and output operations described by some of the standard utilities.

See 2.14.

2.2.2.203 cursor position: The line and column position on the screen denoted by the terminal's cursor.

2.2.2.204 display: To output to the user's terminal.

If the output is not directed to a terminal, the results are undefined.

2.2.2.205 job: A set of processes, comprising a shell pipeline, and any processes descended from it, that are all in the same process group.

See the definition of *pipeline* in 3.9.2.

2.2.2.206 job control job ID: A handle that is used to refer to a job.

The job control job ID can be any of the forms shown in Table 2-100.

Table 2-100 – Job Control Job ID Formats

Job Control Job ID	Meaning
%%	Current job
%+	Current job
%−	Previous job
%*n*	Job number *n*
%*string*	Job whose command begins with *string*
%?*string*	Job whose command contains *string*

2.2.2.207 mount point: Either the root directory or a directory for which the *st_rdev* field of the POSIX.1 {8} *struct stat* differs from that of its parent directory.

2.2.2.208 refresh: To ensure that the information on the user's terminal screen is up-to-date.

2.2.2.209 screen: A rectangular region of columns and lines on a terminal display.

A screen may be a portion of a physical display device or may occupy the entire physical area of the display device.

2.2.2.210 scroll: To move the representation of data vertically or horizontally relative to the terminal screen.

There are two types of scrolling:

(1) The cursor moves with the data

(2) The cursor remains stationary while the data moves

2.2.2.211 suspended job: A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the process group to stop.

A suspended job is a background job, but a background job is not necessarily a suspended job.

2.2.2.212 system scheduling priority: A number used as advice to the system to alter process scheduling priorities.

Raising the value should give a process additional preference when scheduling a process to run. Lowering the value should reduce the preference and make a process less likely to run. Typically, a process with higher system scheduling priority will run to completion more quickly than an equivalent process with lower system scheduling priority. A scheduling priority of zero specifies the default policy of the system.

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

This standard assigns precise requirements for the terms *display* and *write,* unlike some historical user manuals. (The term *write* has been moved into the POSIX.2 base standard as of Draft 11.) Some historical systems have chosen to implement certain utilities without using the traditional UNIX system file descriptor model. For example, the `vi` editor might employ direct screen memory updates on a personal computer, rather than a *write* () system call. An instance of user prompting might appear in a little ‘‘pop-up’’ dialog box, rather than with standard error. When the standard uses the term *display,* the method of outputting to the terminal is unspecified; many historical implementations use *termcap* or *terminfo* , but this is not a requirement. The term *write* is used when the standard mandates that a file descriptor be used and that the output can be redirected. However, it is assumed that when the writing is directly to the terminal (it has not been redirected elsewhere), there is no practical way for a user or test suite to determine whether a file descriptor is actually being used or not. Therefore, the use of a file descriptor is mandated only for the redirection case and the implementation is free to use any method when the output is not redirected. In previous drafts, there were the terms *print* and *type,* which attempted to exert a finer degree of specification on output and input differences for terminals versus other types of files. These verbs were discarded when it was realized that there is no practical or useful way to determine how I/O is accomplished when it is not redirected. *Read* and *write* are used almost exclusively, with the very few exceptions of those utilities where output redirection need not be supported: `tabs`, `talk`, `tput`, and `vi`.

The definition of *block-mode terminal* refers to a mode of operation because some terminals are capable of full character-at-a-time operation only in certain hardware- or software-selected modes. Terminals that can support all the features of this standard in one mode should not be excluded if there are some cases where they may be set up with other characteristics. There is no implication that the implementation is required to sense changes in these modes during operation of the utilities.

The definition of *cursor* is assumed to be a standard in computer science.

The definition of *scroll* omits the situation where the cursor moves while the data remains stationary; this is considered to be a cursor movement, not scrolling.

The definition of *system scheduling priority* is not intended to suggest that all 7
processes in a system have priorities that are comparable. Scheduling policy 7

extensions such as the realtime priorities in POSIX.4 make the notion of a single 7
underlying priority for all scheduling policies problematic. Some systems may 7
implement the `nice`-related features to affect all processes on the system, others 7
to affect just the general time-sharing activities implied by POSIX.2, and others 7
may have no effect at all. Because of the use of ‘‘implementation defined’’ in `nice` 7
and `renice`, a wide range of implementation strategies are possible. 7

2.3 Built-in Utilities

⇒ **2.3 Built-in Utilities.** *Modify the contents of clause 2.3, Built-in Utilities, to add the following utilities to the table of regular built-ins.*

`alias`	`fc`	`jobs`	`unalias`
`bg`	`fg`	`newgrp`	

Built-in Utilities Rationale. *(This subclause is not a part of P1003.2a)*

All of the job-control related utilities are eligible for built-in status because that is the model most historical implementations have used.

2.6 Environment Variables

⇒ **2.6 Environment Variables.** *Modify the contents of clause 2.6, Environment Variables, to add the following variables in the correct sorted order.*

COLUMNS A decimal integer > 0 used to indicate the user's preferred width in column positions for the terminal screen or window. (See the definition of *column position* in 2.2.2.31.) If this variable is unset or null, the implementation determines the number of columns, appropriate for the terminal or window, in an unspecified manner. When **COLUMNS** is set, any terminal-width information implied by **TERM** shall be overridden. Users and conforming applications should not set **COLUMNS** unless they wish to override the system selection and produce output unrelated to the terminal characteristics. This standard specifies the effects of this variable only for the standard utilities defined by the User Portability Utilities Option in Section 5.

LINES A decimal integer > 0 used to indicate the user's preferred number of lines on a page or the vertical screen or window size in lines. A line in this case is a vertical measure large enough to hold the tallest character in the character set being displayed. If this variable is unset or null, the implementation determines the number of lines, appropriate for the terminal or window (size, terminal baud rate, etc.), in an unspecified manner. When **LINES** is set, any terminal-height information implied by **TERM** shall be overridden. Users and conforming applications should not set **LINES** unless they wish to override the system selection and produce output unrelated to the terminal characteristics. This standard specifies the effects of this variable only for the standard utilities defined by the User Portability Utilities Option in Section 5.

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

COLUMNS The default value for the number of column positions is unspecified because historical implementations use different methods to determine values corresponding to the size of the screen in which the utility is run. (In Draft 7, all references to **COLUMNS** and **LINES** were made consistent so that the default is unspecified, rather than implementation defined in a few places. Values for these variables that are not decimal integers > 0 are implicitly undefined values; it is unnecessary to

enumerate all of the possible values outside of the 7
acceptable set.) This size is typically known to the 7
implementation through the value of **TERM**, or by more elaborate methods such as extensions to the `stty` utility, or knowledge of how the user is dynamically resizing windows on a bit-mapped display terminal. Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.

LINES See **COLUMNS**.

2.11.6.1 Standard Output

⇒ **2.11.6.1 Standard Output.** *Modify the contents of subclause 2.11.6.1, Standard Output (part of 2.11, Utility Description Defaults), to add the following paragraph at the end of the subclause:*

When the User Portability Utilities Option is implemented, some of the standard utilities in Section 5 describe their output using the verb *display*, defined in 2.2.2.204. Output described in the Standard Output subclauses of such utilities may be produced using means other than standard output. When standard output is directed to a terminal, the output described shall be written directly to the terminal. Otherwise, the results are undefined.

2.13.2 Symbolic Constants for Portability Specifications

⇒ **2.13.2 Symbolic Constants for Portability Specifications.** *Modify the contents of subclause 2.13.2 to add the following limits in the correct sorted order.*

Name	Description	
{POSIX2_CHAR_TERM}	The system supports at least one terminal type capable of all operations described in this standard. See 2.14. This value need not be present on a system not supporting the User Portability Utilities Option.	7
{POSIX2_UPE}	The system supports the User Portability Utilities Option in Section 5.	

⇒ **B.2.3 Compile-Time Symbolic Constants for Portability Specifications.** *Modify the contents of subclause B.2.3 to add the following limits in the correct sorted order.*

Name	Description	
_POSIX2_CHAR_TERM	The system supports at least one terminal type capable of all operations described in this standard. See 2.14. This value need not be present on a system not supporting the User Portability Utilities Option.	7
_POSIX2_UPE	The system supports the User Portability Utilities Option in Section 5.	

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

See the rationale for {POSIX2_CHAR_TERM} in 2.14.

In an earlier draft, two limits, {POSIX2_MAX_NICE} and {POSIX2_MIN_NICE}, were included to bound the range of acceptable ‘‘nice values.’’ These limits were removed because `nice` and `renice` were reworded to gracefully accept values outside the range without returning an error.

2.14 Terminal Characteristics

⇒ **2.14 Terminal Characteristics.** *Add a new clause 2.14:*

This clause applies only to systems supporting the User Portability Utilities 7
Option. 7

The standard utilities historically have been implemented on a wide range of terminal types, but a conforming implementation need not support all features of all utilities on every conceivable terminal. This standard states which features are optional for certain classes of terminals in the individual utility description clauses. The implementation shall document which terminal types it supports and which of these features and utilities are not supported by each terminal. This implementation-defined list of terminals:

- Shall include at least one terminal type that is capable of supporting all of the standard utilities and all of their features, if the {POSIX2_CHAR_TERM} option is provided.
- May group terminals in terms of families or equivalences to other documented terminal types.
- Need not consist of an exhaustive list of terminal models when the implementor considers that some terminal types are used too infrequently to be listed.

When a feature or utility is not supported on a specific terminal type, as allowed by this standard, and the implementation considers such a condition to be an error preventing use of the feature or utility, the implementation shall indicate such conditions through diagnostic messages or exit status values or both (as appropriate to the specific utility description) that inform the user that the terminal type lacks the appropriate capability.

This standard uses a notational convention based on historical practice that identifies some of the control characters defined in 2.4 in a manner easily remembered by users on many terminals. The correspondence between this ‘‘`control-`*char*’’ notation and the actual control characters is shown in Table 2-101. When this standard refers to a character by its `control-` name, it is referring to the actual control character shown in the Value column of the table, which is not necessarily the exact control key sequence on all terminals. Some terminals have keyboards that do not allow the direct transmission of all the nonalphanumeric characters shown. In such cases, the system documentation shall describe which data sequences transmitted by the terminal shall be interpreted by the system as representing the special characters.

NOTE: The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that the keystrokes represent control-shift-letter sequences.

Table 2-101 – Control Character Names

Name	Value	Name	Value	Name	Value
control-A	<SOH>	control-L	<FF>	control-W	<ETB>
control-B	<STX>	control-M	<CR>	control-X	<CAN>
control-C	<ETX>	control-N	<SO>	control-Y	<EM>
control-D	<EOT>	control-O	<SI>	control-Z	<SUB>
control-E	<ENQ>	control-P	<DLE>	control-[	<ESC>
control-F	<ACK>	control-Q	<DC1>	control-\	<FS>
control-G	<BEL>	control-R	<DC2>	control-]	<GS>
control-H	<BS>	control-S	<DC3>	control-^	<RS>
control-I	<HT>	control-T	<DC4>	control-_	<US>
control-J	<LF>	control-U	<NAK>	control-?	<DEL>
control-K	<VT>	control-V	<SYN>		

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

There are at least two types of terminals that might have difficulty with certain utility features: hard copy terminals and block-mode terminals.

It is the intention of the developers of this standard that the list of supported terminals and features be presented as tersely as the implementors may desire. Subsets of terminal types may be included, based upon marketing requirements. Generous use of correct global statements is preferable to itemized lists of features for each of hundreds of terminal types. Attention should be directed to clarifying nonsupported features on terminal types that would otherwise be expected to support such features, due to those terminals' similarity to documented types.

It is the intention of the developers of this standard that when a terminal cannot support a certain feature, the utility should produce a useful diagnostic status or message explicitly informing the user of this fact. It should not produce a syntax error, ''unknown command,'' or ''unknown feature'' message or status. (This intent is not in the normative text because it is difficult to describe except with examples.)

Terminals that do not have the ability to send some of the special characters are in wide industry use, although not on the historical systems that form the basis for this standard. For example, the 3270 family of mainframe synchronous terminals cannot directly send the control characters. Therefore, implementations have to document how to do so; it could be by using a series of keystrokes, or by special function keys. In the 3270 example, PF3 has been used as a `control-D` equivalent.

The {POSIX2_CHAR_TERM} option was added to allow a vendor supporting only block-mode terminals to still conform, even though some utilities, such as `vi`, have limited usefulness. The `vi` editor is the only utility in POSIX.2 that can be entirely omitted if this option is not supported; others, such as `ex`, `more`, and `talk`, still have useful feature subsets that are unrelated to the terminal variety.

The decision on whether to require conformance to {POSIX2_CHAR_TERM} is strictly one of marketing. Even when this option is provided, however, there is no guarantee that *every* terminal on the system will support all features; the option implies that the software drivers and other support features are installed, but not that any of the terminals were actually hooked up.

Section 3: Revisions to Shell Command Language

⇒ **3.1 Shell Definitions.** *Modify the contents of clause 3.1, Shell Definitions, to add the following definition in the correct sorted order [disregarding the subclause number shown here].*

3.1 Shell Definitions

3.1.101 alias name: A word consisting solely of underscores, digits, and alphabetics from the portable character set (see 2.4) and any of the following characters:

`!` `%` `,` `@` 7

Implementations may allow other characters within alias names as an extension. 7

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

The characters allowed in alias names are from historical practice.

⇒ **3.3.1 Alias Substitution.** *Modify the contents of clause 3.3, Token Recognition, to add the following subclause:*

3.3.1 Alias Substitution

The processing of aliases shall be supported if the system supports the User Portability Utilities Option.

After a token has been delimited, but before applying the grammatical rules in 3.10, a resulting word that is identified to be the command name word of a simple command shall be examined to determine if it is an unquoted, valid alias name. However, reserved words in correct grammatical context shall not be candidates for alias substitution. A valid alias name (see 3.1.101) shall be one that has been defined by the `alias` utility (see 5.1) and not subsequently undefined using `unalias` (see 5.31). Implementations also may provide predefined valid aliases that are in effect when the shell is invoked. To prevent infinite loops in recursive aliasing, if the shell is not currently processing an alias of the same name, the word shall be replaced by the value of the alias; otherwise, it shall not be replaced.

If the value of the alias replacing the word ends in a `<blank>`, the shell shall check the next command word for alias substitution; this process shall continue until a word is found that is not a valid alias or an alias value does not end in a `<blank>`.

When used as specified by this standard, alias definitions shall not be inherited by separate invocations of the shell or by the utility execution environments invoked by the shell; see 3.12.

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

The alias capability was added in the UPE because it is widely used in historical implementations by interactive users. It was omitted from the base standard because it is not commonly used in application scripts, particularly since aliases are not passed to child shells.

The definition of *alias name* precludes an alias name containing a slash character. Since the text applies to the command words of simple commands, reserved words (in their proper places) cannot be confused with aliases.

The placement of alias substitution in Token Recognition implies that it precedes all of the word expansion steps.

An example concerning trailing `<blank>`s and reserved words follows. If the user types:

```
$ alias foo="/bin/ls "
$ alias while="/"
```

The effect of executing

```
$ while true
> do
> echo "Hello, World"
> done
```

is a never-ending sequence of `Hello, World` strings to the screen. However, if the user types

```
$ foo while
```

the result will be an `ls` listing of `/`. Since the alias substitution for `foo` ends in a `<space>`, the next word is checked for alias substitution. The next word, `while`, has also been aliased, so it is substituted as well. Since it is not in the proper position as a command word, it is not recognized as a reserved word.

If the user types

```
$ foo; while
```

`while` retains its normal reserved-word properties.

See the rationale for `alias` for a discussion of exportable aliases.

3.5.3 Variables

⇒ **3.5.3 Variables.** *Modify the contents of subclause 3.5.3, Variables, to add the following environment variables in the correct sorted order.*

ENV This variable, when the shell is invoked, shall be subjected to parameter expansion (see 3.6.2) by the shell and the resulting value shall be used as a pathname of a file containing shell commands to execute in the current environment. The file need not be executable. If the expanded value of **ENV** is not an absolute pathname, the results are unspecified. **ENV** shall be ignored if the user's real and effective user IDs or real and effective group IDs are different. This standard specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

LINENO This variable shall be set by the shell to a decimal number representing the current sequential line number (numbered starting with 1) within a script or function before it executes each command. If the user unsets or resets **LINENO**, the variable may lose its special meaning for the life of the shell. If the shell is not currently executing a script or function, the value of **LINENO** is unspecified. This standard specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

PPID This variable shall be set by the shell to the decimal process ID of the process that invoked this shell. In a subshell (see 3.12), **PPID** shall be set to the same value as that of parent of the current shell. For example, `echo $PPID` and `(echo $PPID)` would produce the same value. This standard specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

PS1 Each time an interactive shell is ready to read a command, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value shall be `"$ "`. For users who have specific additional implementation-defined privileges (see 2.2.2.6) the default may be another, implementation-defined, value. The shell shall replace each instance of the character ! in **PS1** with the history file number of the next command to be typed. (See 5.12.) Escaping the ! with another ! (i.e., !!) shall place the literal character ! in the prompt. This standard specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

PS 2 Each time the user enters a `<newline>` prior to completing a command line in an interactive shell, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value shall be `"> "`. This standard 7
specifies the effects of this variable only for systems supporting 7
the User Portability Utilities Option. 7

PS 4 When an execution trace (`set -x`) is being performed in an interactive shell, before each line in the execution trace, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value shall be `"+ "`. This 7
standard specifies the effects of this variable only for systems 7
supporting the User Portability Utilities Option. 7

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

The following entries represent additional information about variables included in the standard, or rationale for common variables in use by shells that have been excluded.

_ (underscore)
While underscore is historic practice, its overloaded usage in the KornShell is confusing and it has been omitted from POSIX. 2.

ENV This can be used to set aliases and other items local to the invocation of a shell. The file referred to by **ENV** differs from `$HOME/.profile` in that `.profile` is typically executed at session startup, whereas the **ENV** file is executed at the beginning of each shell invocation.

The **ENV** value is interpreted in a manner similar to a dot script, in that the commands are executed in the current environment and the file needs to be readable, but not executable. However, unlike dot scripts, no **PATH** searching is performed. This is used as a guard against Trojan Horse security breaches.

ERRNO Omitted from the standard because the values of POSIX .1 {8} error numbers are not defined in a portable manner.

FCEDIT In an earlier draft, **FCEDIT** was included in this subclause. However, since it affects only the `fc` utility, it has been removed from this more global place. The value of **FCEDIT** does not affect the command-line editing mode in the shell; see the description of 8
`set -o vi` in 3.14.11. 8

LINENO This variable is frequently used in **PS 4** during trace mode (`set -x`).

PPID Without this variable, there is no way for a utility to signal its parent or to find its parent process. This is also useful to know if the shell has been ''orphaned.''