--	--

IEEE P1003.2 Draft 11.2 – September 1991

Copyright © 1991 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only, and subject to the restrictions contained herein.

Permission is hereby also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copies of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

restriction. The command `s/a/A/2047` should be able to substitute the 2047th occurrence of `a` on a line.

4.56 `sh` — Shell, the standard command language interpreter

4.56.1 Synopsis

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

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

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

4.56.2 Description

The `sh` utility is a command language interpreter that shall execute commands read from a command-line string, the standard input, or a specified file. The commands to be executed shall be expressed in the language described in Section 3.

4.56.3 Options

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

The `−a`, `−C`, `−e`, `−f`, `−n`, `−u`, `−v`, and `−x` options are described as part of the `set` utility in 3.14.11. The following additional options shall be supported by the implementation:

`−c` Read commands from the *command_string* operand. Set the value of special parameter `0` (see 3.5.2) from the value of the *command_name* operand and the positional parameters (`$1`, `$2`, etc.) in sequence from the remaining *argument* operands. No commands shall be read from the standard input.

`−i` Specify that the shell is *interactive* ; see below. An implementation may treat specifying the `−i` option as an error if the real user ID of the calling process does not equal the effective user ID or if the real group ID does not equal the effective user ID.

`−s` Read commands from the standard input.

If there are no operands and the `−c` option is not specified, the `−s` option shall be assumed.

If the `−i` option is present, or if there are no operands and the shell's standard input and standard error are attached to a terminal, the shell is considered to be *interactive* . (See 3.1.4.) The behavior of an interactive shell is not fully specified by this standard.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

Implementations may accept the option letters with a leading plus sign (`+`) instead of a leading hyphen (meaning the reverse case of the option as described in this standard). A conforming application shall protect its first operand, if it starts with a plus sign, by preceding it with the `--` argument that denotes ''end of options.''

4.56.4 Operands

The following operands shall be supported by the implementation:

`-` A single hyphen shall be treated as the first operand and then ignored. If both `-` and `--` are given as arguments, or if other operands precede the single hyphen, the results are undefined.

argument The positional parameters (`$1`, `$2`, etc.) shall be set to *arguments* , if any.

command_file

The pathname of a file containing commands. If the pathname contains one or more slash characters, the implementation shall attempt to read that file; the file need not be executable. If the pathname does not contain a slash character:

— The implementation shall attempt to read that file from the current working directory; the file need not be executable.

— If the file is not in the current working directory, the implementation may perform a search for an executable file using the value of **PATH** , as described in 3.9.1.1.

Special parameter `0` (see 3.5.2) shall be set to the value of *command_file* . If `sh` is called using a synopsis form that omits *command_file* , special parameter `0` shall be set to the value of the first argument passed to `sh` from its parent (e.g., *argv* [0] in the C binding), which is normally a pathname used to execute the `sh` utility.

command_name

A string assigned to special parameter `0` when executing the commands in *command_string*. If *command_name* is not specified, special parameter `0` shall be set to the value of the first argument passed to `sh` from its parent (e.g., *argv* [0] in the C binding), which is normally a pathname used to execute the `sh` utility.

command_string

A string that shall be interpreted by the shell as one or more commands, as if the string were the argument to the function in 7.1.1 [such as the *system* () function in the C binding]. If the *command_string* operand is an empty string, `sh` shall exit with a

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

zero exit status.

4.56.5 External Influences

4.56.5.1 Standard Input

The standard input shall be used only if:

(1) The `−s` option is specified, or;

(2) The `−c` option is not specified and no operands are specified, or;

(3) The script executes one or more commands that require input from standard input (such as a `read` command that does not redirect its input).

See Input Files.

When the shell is using standard input and it invokes a command that also uses standard input, the shell shall ensure that the standard input file pointer points directly after the command it has read when the command begins execution. It shall not read ahead in such a manner that any characters intended to be read by the invoked command are consumed by the shell (whether interpreted by the shell or not) or that characters that are not read by the invoked command are not seen by the shell. When the command expecting to read standard input is started asynchronously by an interactive shell, it is unspecified whether characters are read by the command or interpreted by the shell.

If the standard input to `sh` is a FIFO or terminal device and is set to nonblocking reads, then `sh` shall enable blocking reads on standard input. This shall remain in effect when the command completes.

4.56.5.2 Input Files

The input file shall be a text file, except that line lengths shall be unlimited. If the input file is empty or consists solely of blank lines and/or comments, `sh` shall exit with a zero exit status.

4.56.5.3 Environment Variables

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

HOME This variable shall be interpreted as the pathname of the user's home directory. The contents of **HOME** are used in Tilde Expansion as described in 3.6.1.

IFS *Input field separators* : a string treated as a list of characters that shall be used for field splitting and to split lines into words with the `read` command. See 3.6.5. If **IFS** is not set, the shell shall behave as if the value of **IFS** were the `<space>`, `<tab>`, and `<newline>` characters. Implementations may ignore the value of **IFS** in the

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

environment at the time `sh` is invoked, treating **IFS** as if it were not set.

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 behavior of range expressions, equivalence classes, and multicharacter collating elements within pattern matching.

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), which characters are defined as letters (character class `alpha`), and the behavior of character classes within pattern matching.

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

PATH This variable shall represent a string formatted as described in 2.6, used to effect command interpretation. See 3.9.1.1.

4.56.5.4 Asynchronous Events

Default.

4.56.6 External Effects

4.56.6.1 Standard Output

See Standard Error.

4.56.6.2 Standard Error

Except as otherwise stated (by the descriptions of any invoked utilities or in interactive mode), standard error is used only for diagnostic messages.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.56.6.3 Output Files

None.

4.56.7 Extended Description

See Section 3.

4.56.8 Exit Status

The sh utility shall exit with one of the following values: 1

- 0 The script to be executed consisted solely of zero or more blank lines and/or comments. 1 1
- 1–125 A noninteractive shell detected a syntax, redirection, or variable assignment error. 1 1
- 127 A specified *command_file* could not be found by a noninteractive shell. 1 1

Otherwise, the shell shall return the exit status of the last command it invoked or attempted to invoke (see also the exit utility in 3.14.7).

4.56.9 Consequences of Errors

See 3.8.1.

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

Examples, Usage

```
sh -c "cat myfile"

sh my_shell_cmds
```

The sh utility and the set special built-in utility share a common set of options. Unlike set, however, the POSIX.2 sh does not specify the use of + as an option flag, because it is not particularly useful (the + variety generally invokes the default behavior) and because *getopt*() does not support it. However, since many historical implementations do support the plus, applications will have to guard against the relatively obscure case of a first operand with a leading plus sign.

There is a large number of environment variables used by historical implementations of sh that will not be introduced by POSIX.2 until the UPE is completed.

The KornShell ignores the contents of **IFS** upon entry to the script. A conforming application cannot rely on importing **IFS**. One justification for this, beyond security considerations, is to assist possible future shell compilers. Allowing **IFS** to be imported from the environment will prevent many optimizations that might otherwise be performed via dataflow analysis of the script itself.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The standard input and standard error are the files that determine whether a shell is interactive when `−i` is not specified. For example,

`sh > file` and `sh 2> file`

create interactive and noninteractive shells, respectively. Although both accept terminal input, the results of error conditions will be different, as described in 3.8.1; in the second example a redirection error encountered by a special built-in utility will abort the shell.

The text in Standard Input about nonblocking reads concerns an instance of `sh` 1
that has been invoked, probably by a C-language program, with standard input 1
that has been opened using the O_NONBLOCK flag; see POSIX.1 {8} *open*(). If the 1
shell did not reset this flag, it would immediately terminate because no input data 1
would be available yet and that would be considered the same as end-of-file. 1

History of Decisions Made

See the Rationale for Section 3 concerning the lack of interactive features in `sh`. These features, including optional job control, are scheduled to be added in the User Portability Extension.

The **PS1** and **PS2** variables are not specified because this standard, without UPE, does not describe an interactive shell.

The options associated with a *restricted shell* (command name `rsh` and the `−r` option) were excluded because the developers of the standard felt that the implied level of security was not achievable and they did not want to raise false expectations.

On systems that support set-user-ID scripts, a historical trapdoor has been to link a script to the name `−i`. When it is called by a sequence such as `sh -` or by `#! /bin/sh -` the historical systems have assumed that no option letters follow. Thus, POSIX.2 allows the single hyphen to mark the end of the options, in addition to the use of the regular `--` argument, because it was felt that the older practice was so pervasive. An alternative approach is taken by the KornShell, where real and effective user/group IDs must match for an interactive shell; this behavior is specifically allowed by POSIX.2. (Note: there are other problems with set-user-ID scripts that the two approaches described here do not deal with.)

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.57 `sleep` — Suspend execution for an interval

4.57.1 Synopsis

`sleep` *time*

4.57.2 Description

The `sleep` utility shall suspend execution for at least the integral number of seconds specified by the *time* operand.

4.57.3 Options

None.

4.57.4 Operands

The following operands shall be supported by the implementation:

time A nonnegative decimal integer specifying the number of seconds for which to suspend execution.

4.57.5 External Influences

4.57.5.1 Standard Input

None.

4.57.5.2 Input Files

None.

4.57.5.3 Environment Variables

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

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_**.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.