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All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change.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]

3.14 Special Built-in Utilities

The following *special built-in* utilities shall be supported in the shell command language. The output of each command, if any, shall be written to standard output, subject to the normal redirection and piping possible with all commands.

The term *built-in* implies that the shell can execute the utility directly and does not need to search for it. An implementation can choose to make any utility a built-in; however, the special built-in utilities described here differ from regular built-in utilities in two respects:

(1) A syntax error in a special built-in utility may cause a shell executing that utility to abort, while a syntax error in a regular built-in utility shall not cause a shell executing that utility to abort. (See 3.8.1 for the consequences of errors on interactive and noninteractive shells.) If a special built-in utility encountering a syntax error does not abort the shell, its exit value shall be nonzero.

(2) Variable assignments specified with special built-in utilities shall remain in effect after the built-in completes; this shall not be the case with a regular built-in or other utility.

As described in 2.3, the special built-in utilities in this clause need not be provided in a manner accessible via the POSIX .1 {8}*exec* family of functions.

Some of the special built-ins are described as conforming to the utility argument syntax guidelines in 2.10.2. For those that are not, the requirement in 2.11.3 that "--" be recognized as a first argument to be discarded does not apply and a conforming application shall not use that argument.

3.14.1 `break` — Exit from `for`, `while`, or `until` loop

```
break [n]
```

Exit from the smallest enclosing `for`, `while`, or `until` loop, if any; or from the *n* th enclosing loop if *n* is specified. The value of *n* is an unsigned decimal integer ≥ 1. The default is equivalent to *n*=1. If *n* is greater than the number of enclosing loops, the last enclosing loop shall be exited from. Execution continues with the command immediately following the loop.

Exit Status

0 Successful completion.

>0 The *n* value was not an unsigned decimal integer ≥ 1.

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

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

Example :

```
for i in *
do
        if test -d "$i"
        then break
        fi
done
```

Consideration was given to expanding the syntax of the `break` and `continue` to refer to a label associated with the appropriate loop, as a preferable alternative to the [*n*] method. This new method was proposed late in the development of the standard and adequate consensus could not be formed to include it. However, POSIX. 2 does reserve the namespace of command names ending with a colon. It is anticipated that a future implementation could take advantage of this and provide something like:

```
outofloop: for i in a b c d e
do
        for j in 0 1 2 3 4 5 6 7 8 9
        do
                if test -r "${i}${j}"
                then break outofloop
                fi
        done
done
```

and that this might be standardized after implementation experience is achieved.

3.14.2 colon — Null utility

`:` [*argument* ...]

This utility shall only expand command *argument* s.

Exit Status

Zero.

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

The colon (`:`), or null utility, is used when a command is needed, as in the `then` condition of an `if` command, but nothing is to be done by the command.

Example :

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

```
: ${X=abc}
if      false
then    :
else    echo $X
fi
abc
```

As with any of the special built-ins, the null utility can also have variable assignments and redirections associated with it, such as:

```
x=y : > z
```

which sets variable `x` to the value `y` (so that it persists after the null utility ‘‘completes’’) and creates or truncates file `z`.

3.14.3 `continue` — Continue `for`, `while`, or `until` loop

```
continue [n]
```

The `continue` utility shall return to the top of the smallest enclosing `for`, `while`, or `until`, loop, or to the top of the *n*th enclosing loop, if *n* is specified. This involves repeating the condition list of a `while` or `until` loop or performing the next assignment of a `for` loop, and reexecuting the loop if appropriate.

The value of *n* is a decimal integer ≥ 1. The default is equivalent to *n*=`1`. If *n* is greater than the number of enclosing loops, the last enclosing loop is used.

Exit Status

0 Successful completion. 2

>0 The *n* value was not an unsigned decimal integer ≥ 1. 2

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

Example :

```
for i in *
do
        if test –d "$i"
        then continue
        fi
done
```

3.14.4 `dot` — Execute commands in current environment

```
. file
```

The shell shall execute commands from the *file* in the current environment.

If *file* does not contain a slash, the shell shall use the search path specified by **PATH** to find the directory containing *file*. Unlike normal command search, however, the file searched for by the `dot` utility need not be executable. If no

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

readable file is found, a noninteractive shell shall abort; an interactive shell shall write a diagnostic message to standard error, but this condition shall not be considered a syntax error.

Exit Status

Returns the value of the last command executed, or a zero exit status if no command is executed.

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

Some older implementations searched the current directory for the *file*, even if the value of **PATH** disallowed it. This behavior was omitted from POSIX.2 due to concerns about introducing the susceptibility to trojan horses that the user might be trying to avoid by leaving dot out of **PATH**.

The KornShell version of `dot` takes optional arguments that are set to the positional parameters. This is a valid extension that allows a dot script to behave identically to a function.

Example:

```
cat foobar
foo=hello bar=world
. foobar
echo $foo $bar
hello world
```

3.14.5 `eval` — Construct command by concatenating arguments

`eval` [*argument* ...]

The `eval` utility shall construct a command by concatenating *argument*s together, separating each with a <space>. The constructed command shall be read and executed by the shell.

Exit Status

If there are no *argument*s, or only null arguments, `eval` shall return a zero exit status; otherwise, it shall return the exit status of the command defined by the string of concatenated *argument*s separated by spaces.

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

2636 ### 3.14.5.1 `eval` Rationale. *(This subclause is not a part of P1003.2)*

2637 **Example** :

2638 ```
2639 foo=10 x=foo
2640 y='$'$x
2641 echo $y
2642 $foo
2643 eval y='$'$x
2644 echo $y
2645 10
```

2646 ## 3.14.6 `exec` — Execute commands and open, close, and/or copy file descriptors

2647 `exec` [*command* [*argument* ...]]

2648 The `exec` utility opens, closes, and/or copies file descriptors as specified by any
2649 redirections as part of the command.

2650 If `exec` is specified without *command* or *arguments,* and any file descriptors with
2651 numbers > 2 are opened with associated redirection statements, it is unspecified
2652 whether those file descriptors remain open when the shell invokes another utility.

2653 If `exec` is specified with *command* , it shall replace the shell with *command*
2654 without creating a new process. If *argument* s are specified, they are arguments to
2655 *command* . Redirection shall affect the current shell execution environment.

2656 **Exit Status**

2657 If *command* is specified, `exec` shall not return to the shell; rather, the exit status 2
2658 of the process shall be the exit status of the program implementing *command* , 2
2659 which overlaid the shell. If *command* is not found, the exit status shall be 127. If 1
2660 *command* is found, but it is not an executable utility, the exit status shall be 126. 1
2661 If a redirection error occurs (see 3.8.1), the shell shall exit with a value in the 1
2662 range 1–125. Otherwise, `exec` shall return a zero exit status.

2663 ### 3.14.6.1 `exec` Rationale. *(This subclause is not a part of P1003.2)*

2664 Most historical implementations are not conformant in that

2665 ```
foo=bar exec cmd
```

2666 does not pass `foo` to `cmd`.

2667 Earlier drafts stated that ''If specified without *command* or *argument* , the shell
2668 sets to close-on-exec file numbers greater than 2 that are opened in this way, so
2669 that they will be closed when the shell invokes another program.'' This was based
2670 on the behavior of one version of the KornShell and was made unspecified when it
2671 was realized that some existing scripts relied on the more generally historical
2672 behavior (leaving all file descriptors open). Furthermore, since the application
2673 should have no cognizance of whether a new shell is simply *fork* ()ed, rather than

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

exec()ed, it could not consistently rely on the automatic closing behavior anyway. Scripts concerned that child shells could misuse open file descriptors can always close them explicitly, as shown in one of the following examples.

Examples :

Open `readfile` as file descriptor 3 for reading:

```
exec 3< readfile
```

Open `writefile` as file descriptor 4 for writing:

```
exec 4> writefile
```

Make unit 5 a copy of unit 0:

```
exec 5<&0
```

Close file unit 3:

```
exec 3<&–
```

Cat the file `maggie` by replacing the current shell with the `cat` utility:

```
exec cat maggie
```

3.14.7 `exit` — Cause the shell to exit

```
exit [n]
```

The `exit` utility shall cause the shell to exit with the exit status specified by the unsigned decimal integer *n*. If *n* is specified, but its value is not between 0 and 255 inclusively, the exit status is undefined.

A trap on `EXIT` shall be executed before the shell terminates, except when the `exit` utility is invoked in that trap itself, in which case the shell shall exit immediately.

Exit Status

The exit status shall be *n*, if specified. Otherwise, the value shall be the exit value of the last command executed, or zero if no command was executed. When `exit` is executed in a trap action (see 3.14.13), the ‘‘last command’’ is considered to be the command that executed immediately preceding the trap action.

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

As explained in other clauses, certain exit status values have been reserved for special uses and should be used by applications only for those purposes:

126 A file to be executed was found, but it was not an executable utility.

127 A utility to be executed was not found.

>128 A command was interrupted by a signal.

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

Examples :

Exit with a *true* value:

```
exit 0
```

Exit with a *false* value:

```
exit 1
```

3.14.8 `export` — Set export attribute for variables

```
export name [=word ]...
export -p
```

The shell shall give the export attribute to the variables corresponding to the specified *name* s, which shall cause them to be in the environment of subsequently executed commands.

When `-p` is specified, `export` shall write to the standard output the names and values of all exported variables, in the following format:

```
"export %s=%s\n", <name> , <value>
```

The shell shall format the output, including the proper use of quoting, so that it is suitable for re-input to the shell as commands that achieve the same exporting results.

The `export` special built-in shall conform to the utility argument syntax guidelines described in 2.10.2.

Exit Status

Zero.

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

When no arguments are given, the results are unspecified. Some historical shells use the no-argument case as the functional equivalent of what is required here with `-p`. This feature was left unspecified because it is not existing practice in all shells and some scripts may rely on the now-unspecified results on their implementations. Attempts to specify the `-p` output as the default case were unsuccessful in achieving consensus. The `-p` option was added to allow portable access to the values that can be saved and then later restored using, for instance, a dot script.

Examples :

Export **PWD** and **HOME** variables:

```
export PWD HOME
```

Set and export the **PATH** variable:

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

```
export PATH=/local/bin:$PATH
```

Save and restore all exported variables:

```
export -p > temp-file
unset a lot of variables
... processing
. temp-file
```

3.14.9 `readonly` — Set read-only attribute for variables

```
readonly name [=word ]...
readonly -p
```

The variables whose *name* s are specified shall be given the `readonly` attribute. The values of variables with the read-only attribute cannot be changed by subsequent assignment, nor can those variables be unset by the `unset` utility.

When `–p` is specified, `readonly` shall write to the standard output the names and values of all read-only variables, in the following format:

```
"readonly %s=%s\n", <name> , <value>
```

The shell shall format the output, including the proper use of quoting, so that it is suitable for re-input to the shell as commands that achieve the same attribute-setting results.

The `readonly` special built-in shall conform to the utility argument syntax guidelines described in 2.10.2.

Exit Status

Zero.

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

Example :

```
readonly HOME PWD
```

Some versions of the shell exist that preserve the read-only attribute across separate invocations. POSIX. 2 allows this behavior, but does not require it.

See the rationale for `export` (3.14.8.1) for a description of the no-argument and `–p` output cases.

In a previous draft, read-only functions were considered, but they were omitted as not being existing practice or particularly useful. Furthermore, functions must not be `readonly` across invocations to preclude *spoofing* (spoofing is the term for the practice of creating a program that acts like a well-known utility with the intent of subverting the user's real intent) of administrative or security-relevant (or -conscious) shell scripts.

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

3.14.10 `return` — Return from a function

```
return [n]
```

The return utility shall cause the shell to stop executing the current function or dot script (see 3.14.4). If the shell is not currently executing a function or dot script, the results are unspecified.

Exit Status

The value of the special parameter ? shall be set to *n*, an unsigned decimal integer, or to the exit status of the last command executed if *n* is not specified. If the value of *n* is greater than 255, the results are undefined. When `return` is executed in a trap action (see 3.14.13), the ''last command'' is considered to be the command that executed immediately preceding the trap action.

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

The behavior of `return` when not in a function or dot script differs between the System V shell and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is the same as `exit`.

The results of returning a number greater than 255 are undefined because of differing practices in the various historical implementations. Some shells AND out all but the low order 8 bits; others allow larger values, but not of unlimited size.

See the discussion of appropriate exit status values in 3.14.7.1.

3.14.11 `set` — Set/unset options and positional parameters

```
set [-aCefnuvx] [argument ...]
set [+aCefnuvx] [argument ...]
set -- [argument ...]
```

Obsolescent version :

```
set - [argument ...]
```

If no options or *argument* s are specified, `set` shall write the names and values of all shell variables in the collation sequence of the current locale. Each *name* shall start on a separate line, using the format:

```
"%s=%s\n", <name> , <value>
```

The *value* string shall be written with appropriate quoting so that it is suitable for re-input to the shell, (re)setting, as far as possible, the variables that are currently set. Readonly variables cannot be reset. See the description of shell quoting in 3.2.

When options are specified, they shall set or unset attributes of the shell, as described below. When *argument* s are specified, they shall cause positional parameters to be set or unset, as described below. Setting/unsetting attributes

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

and positional parameters are not necessarily related actions, but they can be combined in a single invocation of `set`.

The `set` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that options can be specified with either a leading hyphen (meaning enable the option) or plus-sign (meaning disable it).

The implementation shall support the options in the following list in both their hyphen and plus-sign forms. These options can also be specified as options to `sh`; see 4.56.

`-a` When this option is on, the export attribute shall be set for each variable to which an assignment is performed. (See 3.1.15.) If the assignment precedes a utility name in a command, the export attributes shall not persist in the current execution environment after the utility completes, with the exception that preceding one of the special built-in utilities shall cause the export attribute to persist after the built-in has completed. If the assignment does not precede a utility name in the command, or if the assignment is a result of the operation of the `getopts` or `read` utilities (see 4.27 and 4.52), the export attribute shall persist until the variable is unset.

`-C` (Uppercase C.) Prevent existing files from being overwritten by the shell's `>` redirection operator (see 3.7.2); the `>|` redirection operator shall override this ''noclobber'' option for an individual file.

`-e` When this option is on, if a simple command fails for any of the reasons listed in 3.8.1 or returns an exit status value >0, and is not part of the compound list following a `while`, `until`, or `if` keyword, and is not a part of an AND or OR list, and is not a pipeline preceded by the `!` reserved word, then the shell immediately shall exit.

`-f` The shell shall disable pathname expansion.

`-n` The shell shall read commands but not execute them; this can be used to check for shell script syntax errors. An interactive shell may ignore this option.

`-u` The shell shall write a message to standard error when it tries to expand a variable that is not set and immediately exit. An interactive shell shall not exit.

`-v` The shell shall write its input to standard error as it is read.

`-x` The shell shall write to standard error a trace for each command after it expands the command and before it executes it.

The default for all these options is off (unset) unless the shell was invoked with them on (see `sh` in 4.56). All the positional parameters shall be unset before any new values are assigned.

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

The remaining arguments shall be assigned in order to the positional parameters. The special parameter # shall be set to reflect the number of positional parameters.

The special argument `"--"` immediately following the `set` command name can be used to delimit the arguments if the first argument begins with `+` or `−`, or to prevent inadvertent listing of all shell variables when there are no arguments. The command `set −−` without *argument* s shall unset all positional parameters and set the special parameter # to zero.

In the obsolescent version, the `set` command name followed by − with no other arguments shall turn off the `−v` and `−x` options without changing the positional parameters. The `set` command name followed by − with other arguments shall turn off the `−v` and `−x` options and assign the arguments to the positional parameters in order.

Exit Status

Zero.

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

The `set −−` form is listed specifically in the Synopsis even though this usage is implied by the utility syntax guidelines. The explanation of this feature removes any ambiguity about whether the `set −−` form might be misinterpreted as being equivalent to `set` without any options or arguments. The functionality of this form has been adopted from the KornShell. In System V, `set −−` only unsets parameters if there is at least one argument; the only way to unset all parameters is to use `shift`. Using the KornShell version should not affect System V scripts because there should be no reason to deliberately issue it without arguments; if it were issued as, say:

```
set −− "$@"
```

and there were in fact no arguments resulting from `$@`, unsetting the parameters would be a no-op anyway.

The `set +` form in earlier drafts was omitted as being an unnecessary duplication of `set` alone and not widespread historical practice.

The noclobber option was changed to `−C` from the `set -o noclobber` option in previous drafts. The `set −o` is used in the KornShell to accept word-length option names, duplicating many of the single-letter names. The noclobber option was changed to a single letter so that the historical `$-` paradigm would not be broken; see 3.5.2.

The following `set` flags were intentionally omitted with the following rationale:

`−h` This flag is related to command name hashing, which is not required for an implementation. It is primarily a performance issue, which is outside the scope of this standard.

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

2894 `-k` The `-k` flag was originally added by Bourne to make it easier for users of
2895 prerelease versions of the shell. In early versions of the Bourne shell
2896 the construct `set name=value`, had to be used to assign values to shell
2897 variables. The problem with `-k` is that the behavior affects parsing, vir-
2898 tually precluding writing any compilers. To explain the behavior of `-k`,
2899 it is necessary to describe the parsing algorithm, which is implementa-
2900 tion defined. For example,

2901 ```
set -k; echo name=value
```

2902 and

2903 ```
set -k
2904 echo name=value
```

2905 behave differently. The interaction with functions is even more com-
2906 plex. What is more, the `-k` flag is never needed, since the command line
2907 could have been reordered.

2908 `-t` The `-t` flag is hard to specify and almost never used. The only known
2909 use could be done with here-documents. Moreover, the behavior with
2910 `ksh` and `sh` differ. The man page says that it exits after reading and
2911 executing one command. What is one command? If the input is
2912 `date;date`, `sh` executes both `date` commands, `ksh` does only the first.

2913 Consideration was given to rewriting `set` to simplify its confusing syntax. A
2914 specific suggestion was that the `unset` utility should be used to unset options
2915 instead of using the non-*getopt* ()-able *+option* syntax. However, the conclusion
2916 was reached that people were satisfied with the existing practice of using *+option*
2917 and there was no compelling reason to modify such widespread existing practice.

2918 **Examples** :

2919 Write out all variables and their values:

2920 ```
set
```

2921 Set `$1`, `$2`, and `$3` and set `$#` to 3:

2922 ```
set c a b
```

2923 Turn on the `-x` and `-v` options:

2924 ```
set -xv
```

2925 Unset all positional parameters:

2926 ```
set --
```

2927 Set `$1` to the value of `x`, even if `x` begins with `-` or `+`:

2928 ```
set -- "$x"
```

2929 Set the positional parameters to the expansion of `x`, even if `x` expands with a lead-
2930 ing `-` or `+`:

2931 ```
set -- $x
```

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

3.14.12 `shift` — Shift positional parameters

```
shift [n]
```

The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so forth. The parameters represented by the numbers `$#` down to `$#`–*n*+1 shall be unset, and the parameter `#` shall be updated to reflect the new number of positional parameters.

The value *n* shall be an unsigned decimal integer less than or equal to the value of the special parameter `#`. If *n* is not given, it shall be assumed to be `1`. If *n* is `0`, the positional and special parameters shall not be changed.

Exit Status

The exit status shall be >0 if *n*>`$#`; otherwise, it shall be zero.

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

Example :

```
set a b c d e
shift 2
echo $*
c d e
```

3.14.13 `trap` — Trap signals

```
trap [action condition ...]
```

If *action* is –, the shell shall reset each *condition* to the default value. If *action* is null (`''`), the shell shall ignore each of the specified *condition* s if they arise. Otherwise, the argument *action* shall be read and executed by the shell when one of the corresponding conditions arises. The action of the trap shall override a previous action (either default action or one explicitly set). The value of `$?` after the trap action completes shall be the value it had before the trap was invoked.

The condition can be `EXIT`, `0` (equivalent to `EXIT`), or a signal specified using a symbolic name, without the `SIG` prefix, as listed in Required Signals and Job Control Signals (Table 3-1 and Table 3-2 in POSIX .1 {8}). (For example: `HUP`, `INT`, `QUIT`, `TERM`). Setting a trap for SIGKILL or SIGSTOP produces undefined results.

The environment in which the shell executes a trap on `EXIT` shall be identical to the environment immediately after the last command executed before the trap on `EXIT` was taken.

Each time the trap is invoked, the *action* argument shall be processed in a manner equivalent to:

```
eval "$action"
```

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

Signals that were ignored on entry to a noninteractive shell cannot be trapped or reset, although no error need be reported when attempting to do so. An interactive shell may reset or catch signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed with another `trap` command.

The `trap` command with no arguments shall write to standard output a list of commands associated with each condition. The format is:

`"trap -- %s %s ...\n"`, *<action>*, *<condition>* ...

The shell shall format the output, including the proper use of quoting, so that it is suitable for re-input to the shell as commands that achieve the same trapping results.

An implementation may allow numeric signal numbers for the conditions as an extension, if and only if the following map of signal numbers to names is true:

Signal Number	Signal Name	Signal Number	Signal Name
1	SIGHUP	9	SIGKILL
2	SIGINT	14	SIGALRM
3	SIGQUIT	15	SIGTERM
6	SIGABRT		

Otherwise, it shall be an error for the application to use numeric signal numbers.

The `trap` special built-in shall conform to the utility argument syntax guidelines described in 2.10.2.

Exit Status

If the trap name or number is invalid, a nonzero exit status shall be returned; otherwise, zero shall be returned. For both interactive and noninteractive shells, invalid signal names or numbers shall not be considered a syntax error and shall not cause the shell to abort.

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

Implementations may permit lowercase signal names as an extension. Implementations may also accept the names with the `SIG` prefix; no known historical shell does so. The `trap` and `kill` utilities in POSIX.2 are now consistent in their omission of the `SIG` prefix for signal names. Some `kill` implementations do not allow the prefix and `kill -l` lists the signals without prefixes.

As stated previously, when a subshell is entered, traps are set to the default actions. This does not imply that the `trap` command cannot be used within the subshell to set new traps.

Trapping SIGKILL or SIGSTOP is accepted by some historical implementations, but it does not work. Portable POSIX.2 applications cannot try it.

The output format is not historical practice. Since the output of historical `trap`s is not portable (because numeric signal values are not portable) and had to

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

change to become so, an opportunity was taken to format the output in a way that a shell script could use to save and then later reuse a trap if it wanted. For example:

```
save_traps=$(trap)
...
eval "$save_traps"
```

The KornShell uses an `ERR` trap that is triggered whenever `set -e` would cause an exit. This is allowable as an extension, but was not mandated, as other shells have not used it.

The text about the environment for the `EXIT` trap invalidates the behavior of some historical versions of interactive shells which, e.g., close the standard input before executing a trap on 0. For example, in some historical interactive shell sessions the following trap on 0 would always print `--`:

```
trap 'read foo; echo "-$foo-"' 0
```

Examples :

Write out a list of all traps and actions:

```
trap
```

Set a trap so the `logout` utility in the **HOME** directory will execute when the shell terminates:

```
trap '$HOME/logout' EXIT
```

or

```
trap '$HOME/logout' 0
```

Unset traps on INT , QUIT , TERM , and EXIT :

```
trap - INT QUIT TERM EXIT
```

3.14.14 `unset` — Unset values and attributes of variables and functions

`unset [-fv]` *name* ... 1

Each variable or function specified by *name* shall be unset.

If `-v` is specified, *name* refers to a variable name and the shell shall unset it and remove it from the environment. Read-only variables cannot be unset. 1 1

If `-f` is specified, *name* refers to a function and the shell shall unset the function definition. 1 1

If neither `-f` nor `-v` is specified, *name* refers to a variable; if a variable by that name does not exist, it is unspecified whether a function by that name, if any, shall be unset. 1 1 1

Unsetting a variable or function that was not previously set shall not be considered an error and shall not cause the shell to abort. 1

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

The `unset` special built-in shall conform to the utility argument syntax guidelines described in 2.10.2.

Exit Status

0 All *name* s were successfully unset.

>0 At least one *name* could not be unset.

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

Note that

```
VARIABLE=
```

is not equivalent to an `unset` of `VARIABLE`; in the example, `VARIABLE` is set to "". Also, the ''variables'' that can be `unset` should not be misinterpreted to include the special parameters (see 3.5.2).

Consideration was given to omitting the `-f` option in favor of an `unfunction` utility, but decided to retain existing practice.

The `-v` option was introduced because System V historically used one name space for both variables and functions. When `unset` is used without options, System V historically unset either a function or a variable and there was no confusion about which one was intended. A portable POSIX.2 application can use `unset` without an option to unset a variable, but not a function; the `-f` option must be used.

Examples :

Unset the **VISUAL** variable:

```
unset -v VISUAL
```

Unset the functions `foo` and `bar`:

```
unset -f foo bar
```

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

Section 4: Execution Environment Utilities

The Execution Environment Utilities are the utilities that shall be implemented in all conforming POSIX.2 systems.

4.1 awk — Pattern scanning and processing language

4.1.1 Synopsis

`awk` [`-F` *ERE*] [`-v` *assignment*] ... *program* [*argument* ...]

`awk` [`-F` *ERE*] `-f` *progfile* ... [`-v` *assignment*] ... [*argument* ...]

4.1.2 Description

The `awk` utility shall execute programs written in the *awk* programming language, which is specialized for textual data manipulation. An `awk` program is a sequence of patterns and corresponding actions. When input is read that matches a pattern, the action associated with that pattern shall be carried out.

Input shall be interpreted as a sequence of records. By default, a record is a line, but this can be changed by using the `RS` built-in variable. Each record of input shall be matched in turn against each pattern in the program. For each pattern matched, the associated action shall be executed.

The `awk` utility shall interpret each input record as a sequence of fields where, by default, a field is a string of non-`<blank>` characters. This default white space field delimiter can be changed by using the `FS` built-in variable or the `-F` *ERE*. The `awk` utility shall denote the first field in a record `$1`, the second `$2`, and so forth. The symbol `$0` shall refer to the entire record; setting any other field shall cause the reevaluation of `$0`. Assigning to `$0` shall reset the values of all other fields and the `NF` built-in variable.

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