IEEE P1003.2 Draft 11.2 – September 1991

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]

3.5 Parameters and Variables

A parameter can be denoted by a name, a number, or one of the special characters listed in 3.5.2. A variable is a parameter denoted by a name.

A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can only be unset by using the `unset` special built-in command.

3.5.1 Positional Parameters

A positional parameter is a parameter denoted by the decimal value represented by one or more digits, other than the single digit 0. When a positional parameter with more than one digit is specified, the application shall enclose the digits in braces (see 3.6.2). Positional parameters are initially assigned when the shell is invoked (see `sh` in 4.56), temporarily replaced when a shell function is invoked (see 3.9.5), and can be reassigned with the `set` special built-in command.

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

The digits denoting the positional parameters are always interpreted as a decimal value, even if there is a leading zero.

3.5.2 Special Parameters

Listed below are the special parameters and the values to which they shall expand. Only the values of the special parameters are listed; see 3.6 for a detailed summary of all the stages involved in expanding words.

* Expands to the positional parameters, starting from one. When the expansion occurs within a double-quoted string (see 3.2.3), it expands to a single field with the value of each parameter separated by the first character of the **IFS** variable, or by a `<space>` if **IFS** is unset.

@ Expands to the positional parameters, starting from one. When the expansion occurs within double-quotes, each positional parameter expands as a separate field, with the provision that the expansion of the first parameter is still joined with the beginning part of the original word (assuming that the expanded parameter was embedded within a word), and the expansion of the last parameter is still joined with the last part of the original word. If there are no positional parameters, the 1
expansion of @ shall generate zero fields, even when @ is double-quoted. 1

Expands to the decimal number of positional parameters.

? Expands to the decimal exit status of the most recent pipeline (see 3.9.2).

– (Hyphen) Expands to the current option flags (the single-letter option names concatenated into a string) as specified on invocation, by the `set` special built-in command, or implicitly by the shell.

`$` Expands to the decimal process ID of the invoked shell. In a subshell (see 3.12), `$` shall expand to the same value as that of the current shell.

`!` Expands to the decimal process ID of the most recent background command (see 3.9.3) executed from the current shell. For a pipeline, the process ID is that of the last command in the pipeline.

`0` (Zero.) Expands to the name of the shell or shell script. See `sh` (4.56) for a detailed description of how this name is derived.

See the description of the **IFS** variable in 3.5.3.

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

Most historical implementations implement subshells by forking; thus, the special parameter `$` does not necessarily represent the process ID of the shell process executing the commands since the subshell execution environment preserves the value of `$`.

If a subshell were to execute a background command, the value of its parent's `$!` would not change. For example:

```
(
date &
echo $!
)
echo $!
```

would echo two different values for `$!`.

The descriptions of parameters * and @ assume the reader is familiar with the field splitting discussion in 3.6.5 and understands that portions of the word will remain concatenated unless there is some reason to split them into separate fields. Some examples of the * and @ properties, including the concatenation aspects:

```
set "abc" "def ghi" "jkl"

echo $*         => "abc" "def" "ghi" "jkl"
echo "$*"       => "abc def ghi jkl"
echo $@         => "abc" "def" "ghi" "jkl"
```

but

```
echo "$@"       => "abc" "def ghi" "jkl"
echo "xx$@yy"   => "xxabc" "def ghi" "jklyy"
echo "$@$@"     => "abc" "def ghi" "jklabc" "def ghi" "jkl"
```

In the preceding examples, the double-quote characters that appear after the `=>` do not appear in the output and are used only to illustrate word boundaries.

Historical versions of the Bourne shell have used `<space>` as a separator between the expanded members of `"$*"`. The KornShell has used the first character in **IFS**, which is `<space>` by default. If **IFS** is set to a null string, this is not equivalent to unsetting it; its first character will not exist, so the parameter values are concatenated. For example:

```
$ IFS=''
$ set foo bar bam
$ echo "$@"
foo bar bam
$ echo "$*"
foobarbam
$ unset IFS
$ echo "$*"
foo bar bam
```

The `$-` can be used to save and restore `set` options:

```
Save=$(echo $- | sed 's/[ics]//g')
...
set +aCefnuvx
set -$Save
```

The three options are removed using `sed` in the example because they may appear in the value of `$-` (from the `sh` command line), but are not valid options to `set`.

The command name (parameter `0`) is not counted in the number given by # because it is a special parameter, not a positional parameter.

3.5.3 Variables

Variables shall be initialized from the environment (as defined by POSIX.1 {8}) and can be given new values with variable assignment commands. If a variable is initialized from the environment, it shall be marked for export immediately; see 3.14.8. New variables can be defined and initialized with variable assignments, with the `read` or `getopts` utilities, with the *name* parameter in a `for` loop (see 3.9.4.2), with the `${`*name* `=`*word* `}` expansion, or with other mechanisms provided as implementation extensions. The following variables shall affect the execution of the shell:

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

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

LANG This variable shall provide a default value for the **LC_*** variables, as described in 2.6.

LC_ALL This variable shall interact with the **LANG** and **LC_*** variables as described in 2.6.

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 interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters), 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 represents a string formatted as described in 2.6, used to effect command interpretation. See 3.9.1.1.

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

A description of **PWD** (which is automatically set by the KornShell whenever the current working directory changes) was omitted because its functionality is easily reproduced using `$(pwd)`.

See the discussion of **IFS** in 3.6.5.1.

Other common environment variables used by historical shells are not specified by this standard, but they should be reserved for the historical uses. For interactive use, other shell variables are expected to be introduced by the UPE (and this rationale will be updated accordingly): **ENV**, **FCEDIT**, **HISTFILE**, **HISTSIZE**, **LINENO**, **PPID**, **PS 1**, **PS 2**, **PS 4**.

Tilde expansion for components of the **PATH** in an assignment such as:

```
PATH=~hlj/bin:~dwc/bin:$PATH
```

is a feature of some historical shells and is allowed by the wording of 3.6.1. Note that the tildes are expanded during the assignment to **PATH**, not when **PATH** is accessed during command search.

3.6 Word Expansions

This clause describes the various expansions that are performed on words. Not all expansions are performed on every word, as explained in the following subclauses.

Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and quote removals that occur within a single word expand to a single field. It is only field splitting or pathname expansion that can create multiple fields from a single word. The single exception to this rule is the expansion of the special parameter @ within double-quotes, as is described in 3.5.2.

The order of word expansion shall be as follows: