-- --

IEEE P1003.2 Draft 11.2 – September 1991

4.27 **getopts** — Parse utility options

4.27.1 Synopsis

`getopts` *optstring* *name* [*arg* . . .]

4.27.2 Description

The `getopts` utility can be used to retrieve options and option-arguments from a list of parameters. It shall support the utility argument syntax guidelines 3 through 10, inclusive, described in 2.10.2.

Each time it is invoked, the `getopts` utility shall place the value of the next option in the shell variable specified by the *name* operand and the index of the next argument to be processed in the shell variable **OPTIND** . Whenever the shell is invoked, **OPTIND** shall be initialized to 1.

When the option requires an option-argument, the `getopts` utility shall place it in the shell variable **OPTARG** . If no option was found, or if the option that was found does not have an option-argument, **OPTARG** shall be unset. 1

If an option character not contained in the *optstring* operand is found where an option character is expected, the shell variable specified by *name* shall be set to the question-mark (?) character. In this case, if the first character in *optstring* is a colon (`:`), the shell variable **OPTARG** shall be set to the option character found, but no output shall be written to standard error; otherwise, the shell variable **OPTARG** shall be unset and a diagnostic message shall be written to standard error. This condition shall be considered to be an error detected in the way arguments were presented to the invoking application, but shall not be an error in `getopts` processing.

If an option-argument is missing:

— If the first character of *optstring* is a colon, the shell variable specified by *name* shall be set to the colon character and the shell variable **OPTARG** shall be set to the option character found.

— Otherwise, the shell variable specified by *name* shall be set to the question-mark character, the shell variable **OPTARG** shall be unset, and a diagnostic message shall be written to standard error. This condition shall be considered to be an error detected in the way arguments were presented to the invoking application, but shall not be an error in `getopts` processing; a diagnostic message shall be written as stated, but the exit status shall be zero.

When the end of options is encountered, the `getopts` utility shall exit with a return value greater than zero; the shell variable **OPTIND** shall be set to the index of the first nonoption-argument, where the first `--` argument is considered to be an option-argument if there are no other nonoption-arguments appearing before it, or the value `$#` + 1 if there are no nonoption-arguments; the *name* variable shall be set to the question-mark character. Any of the following shall identify the end of options: the special option `--`, finding an argument that does not begin with a `-`, or encountering an error.

The shell variables **OPTIND** and **OPTARG** shall be local to the caller of `getopts` and shall not be exported by default.

The shell variable specified by the *name* operand, **OPTIND** , and **OPTARG** shall affect the current shell execution environment; see 3.12.

If the application sets **OPTIND** to the value 1, a new set of parameters can be 1
used: either the current positional parameters or new *arg* values. Any other 1
attempt to invoke `getopts` multiple times in a single shell execution environ- 1
ment with parameters (positional parameters or *arg* operands) that are not the 1
same in all invocations, or with an **OPTIND** value modified to be a value other 1
than 1, produces unspecified results. 1

4.27.3 Options

None.

4.27.4 Operands

The following operands shall be supported by the implementation:

optstring A string containing the option characters recognized by the utility invoking `getopts`. If a character is followed by a colon, the option shall be expected to have an argument, which should be supplied as a separate argument. Applications should specify an option character and its option-argument as separate arguments, but `getopts` shall interpret the characters following an option character requiring arguments as an argument whether or not this is done. An explicit null option-argument need not be recognized if it is not supplied as a separate argument when `getopts` is invoked. [See also the *getopt* () Description in B.7]. The characters question-mark and colon shall not be used as option charac-
ters by an application. The use of other option characters that 2
are not alphanumeric produces unspecified results. If the option- 2
argument is not supplied as a separate argument from the option character, the value in **OPTARG** shall be stripped of the option character and the `'-'`. The first character in *optstring* shall determine how `getopts` shall behave if an option character is not known or an option-argument is missing. See 4.27.2.

name The name of a shell variable that shall be set by the `getopts` utility to the option character that was found. See 4.27.2.

The `getopts` utility by default shall parse positional parameters passed to the invoking shell procedure. If *arg* s are given, they shall be parsed instead of the positional parameters.

4.27.5 External Influences

4.27.5.1 Standard Input

None.

4.27.5.2 Input Files

None.

4.27.5.3 Environment Variables

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

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_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).

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

OPTIND This variable shall be used by the `getopts` utility as the index of the next argument to be processed.

4.27.5.4 Asynchronous Events

Default.

4.27.6 External Effects

4.27.6.1 Standard Output

None.

4.27.6.2 Standard Error

Whenever an error is detected and the first character in the *optstring* operand is not a colon (`:`), a diagnostic message shall be written to standard error with the following information in an unspecified format:

— The invoking program name shall be identified in the message. The invoking program name shall be the value of the shell special parameter `0` (see 3.5.2) at the time the `getopts` utility is invoked. A name equivalent to

```
basename "$0"
```

may be used.

— If an option is found that was not specified in *optstring,* this error shall be identified and the invalid option character shall be identified in the message.

— If an option requiring an option-argument is found, but an option-argument is not found, this error shall be identified and the invalid option character shall be identified in the message.

4.27.6.3 Output Files

None.

4.27.7 Extended Description

None.

4.27.8 Exit Status

The `getopts` utility shall exit with one of the following values:

0 An option, specified or unspecified by *optstring* , was found.

>0 The end of options was encountered or an error occurred.

4.27.9 Consequences of Errors

Default.

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

Examples, Usage

The `getopts` utility was chosen in preference to the `getopt` utility specified in System V because `getopts` handles option-arguments containing `<blank>` characters.

Since `getopts` affects the current shell execution environment, it is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
    (getopts abc value "$@")                                        1
    nohup getopts ...                                               1
    find . -exec getopts ... \;                                     1
```

it will not affect the shell variables in the caller's environment. 1

Note that shell functions share **OPTIND** with the calling shell even though the positional parameters are changed. Functions that want to use `getopts` to parse their arguments will usually want to save the value of **OPTIND** on entry and restore it before returning. However, there will be cases when a function will want to change **OPTIND** for the calling shell.

The following example script parses and displays its arguments:

```
aflag=
bflag=
while getopts ab: name
do
        case $name in
        a)      aflag=1;;
        b)      bflag=1
                bval="$OPTARG";;
        ?)      printf "Usage: %s: [-a] [-b value] args\n" $0      1
                exit 2;;
        esac
done
if [ ! -z "$aflag" ]; then                                         1
        printf "Option -a specified\n"                             1
fi                                                                 1
if [ ! -z "$bflag" ]; then                                         1
        printf 'Option -b "%s" specified\n' "$bval"                1
fi                                                                 1
shift $(($OPTIND - 1))                                             1
printf "Remaining arguments are: %s\n" "$*"                        1
```

History of Decisions Made

The **OPTARG** variable is not mentioned in the Environment Variables subclause because it does not affect the execution of `getopts`; it is one of the few ‘‘output-only’’ variables used by the standard utilities.

Use of colon (`:`) as an option character (in a previous draft) was new behavior and violated the syntax guidelines. Many objectors felt that it did not add enough to `getopts` to warrant mandating the extension to existing practice. The colon is now specified to behave as in the KornShell version of the `getopts` utility; when used as the first character in the *optstring* operand, it disables diagnostics concerning missing option-arguments and unexpected option characters. This replaces the use of the **OPTERR** variable that was specified in an earlier draft.

The formats of the diagnostic messages produced by the `getopts` utility and the 1
getopt () function are not fully specified because implementations with superior 1
(‘‘friendlier’’) formats objected to the formats used by some historical implementa- 1
tions. It was felt to be important that the information in the messages used be 1

uniform between `getopts` and *getopt* (). Exact duplication of the messages might not be possible, particularly if a utility is built on another system that has a different *getopt* () function, but the messages must have specific information included so that the program name, invalid option character, and type of error can be distinguished by a user.

Only a rare application program will intercept a `getopts` standard error message and want to parse it. Therefore, implementations are free to choose the most usable messages they can devise. The following formats are used by many historical implementations:

> `"%s: illegal option -- %c\n"`, *<program name>* , *<option character>*

> `"%s: option requires an argument -- %c\n"`, *<program name>* , *<option character>*

Historical shells with built-in versions of *getopt* () or `getopts` have used different formats, frequently not even indicating the option character found in error.

4.28 `grep` — File pattern searcher

4.28.1 Synopsis

`grep [ −E | −F ] [ −c | −l | −q ] [−insvx] −e` *pattern_list* ... `[−f` *pattern_file* `]` ... [*file* ...]

`grep [ −E | −F ] [ −c | −l | −q ] [−insvx] [−e` *pattern_list* `]` ... `−f` *pattern_file* ... [*file* ...]

`grep [ −E | −F ] [ −c | −l | −q ] [−insvx]` *pattern_list* [*file* ...]

Obsolescent Versions :

`egrep [ −c | −l ] [−inv] −e` *pattern_list* [*file* ...]

`egrep [ −c | −l ] [−inv] −f` *pattern_file* [*file* ...]

`egrep [ −c | −l ] [−inv]` *pattern_list* [*file* ...]

`fgrep [ −c | −l ] [−invx] −e` *pattern_list* [*file* ...]

`fgrep [ −c | −l ] [−invx] −f` *pattern_file* [*file* ...]

`fgrep [ −c | −l ] [−invx]` *pattern_list* [*file* ...]