Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'getopts.1p' command

$ man getopts.1p

GETOPTS(1P) POSIX Programmer's Manual GETOPTS(1P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getopts ? parse utility options

SYNOPSIS

getopts optstring name [arg...]

DESCRIPTION

The getopts utility shall retrieve options and option-arguments from a list of parameters. It shall support the Utility Syntax Guidelines 3 to 10, inclusive, described in the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines.

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.

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 out? put 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 de? tected 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 vari? able 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 un? set, 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 operand, or the value "$#"+1 if there are no operands; the name variable shall be set to the <question-mark> character. Any of the following shall identify the end of options: the first "--" argument that is not an option-argument, finding an argument that is not an option-argument and does not begin with a '-', or en? countering 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 Section 2.12, Shell Execution Environment.

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

OPTIONS

None.

OPERANDS

The following operands shall be supported:

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 charac?ters following an option character requiring arguments as an argument whether or not this is done. An explicit null op?tion-argument need not be recognized if it is not supplied as a separate argument when getopts is invoked. (See also the getopt() function defined in the System Interfaces volume of POSIX.1?2017.) The characters <question-mark> and <colon> shall not be used as option characters by an application. The use of other option characters that are not alphanumeric pro?duces unspecified results. If the option-argument is not sup?plied 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 determines how getopts behaves if an option character is not known or an op?

tion-argument is missing.

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

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

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of getopts:

LANG Provide a default value for the internationalization vari? ables that are unset or null. (See the Base Definitions vol? ume of POSIX.1?2017, Section 8.2, Internationalization Vari? ables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard er? ror.

NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.

OPTIND This variable shall be used by the getopts utility as the in? dex of the next argument to be processed.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

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 for? mat:

* The invoking program name shall be identified in the message. The invoking program name shall be the value of the shell special pa? rameter 0 (see Section 2.5.2, Special Parameters) 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 er? ror is identified and the invalid option character shall be identi? fied 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 in? valid option character shall be identified in the message.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

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

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

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

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 "$@")
nohup getopts ...
find . -exec getopts ... \;
```

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

Note that shell functions share OPTIND with the calling shell even though the positional parameters are changed. If the calling shell and any of its functions uses getopts to parse arguments, the results are unspecified.

EXAMPLES

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" $0
          exit 2;;
    esac
done
if [ ! -z "$aflag" ]; then
    printf "Option -a specified\n"
fi
if [ ! -z "$bflag" ]; then
    printf 'Option -b "%s" specified\n' "$bval"
fi
shift $(($OPTIND - 1))
printf "Remaining arguments are: %s\n$*"
```

RATIONALE

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

The OPTARG variable is not mentioned in the ENVIRONMENT VARIABLES sec? tion because it does not affect the execution of getopts; it is one of the few ``output-only'' variables used by the standard utilities.

The <colon> is not allowed as an option character because that is not historical behavior, and it violates the Utility Syntax Guidelines. 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 op? erand, it disables diagnostics concerning missing option-arguments and unexpected option characters. This replaces the use of the OPTERR vari? able that was specified in an early proposal.

The formats of the diagnostic messages produced by the getopts utility and the getopt() function are not fully specified because implementa? tions with superior (``friendlier'') formats objected to the formats used by some historical implementations. The standard developers con? sidered it important that the information in the messages used be uni? form 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 intercepts a getopts standard error message and wants 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 char?

acter found in error.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5.2, Special Parameters

The Base Definitions volume of POSIX.1?2017, Chapter 8, Environment Variables, Section 12.2, Utility Syntax Guidelines

The System Interfaces volume of POSIX.1?2017, getopt()

COPYRIGHT