NAME

getopts – parse utility options

SYNOPSIS

/usr/bin/getopts *optstring name* [*arg*. . .]

sh

getopts *optstring name* [*argument* . . .]

ksh

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

DESCRIPTION

/usr/bin/getopts

The **getopts** utility can be used to retrieve options and option-arguments from a list of parameters.

Each time it is invoked, the **getopts** utility places 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** will be initialised to 1.

When the option requires an option-argument, the **getopts** utility will 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** will 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* will be set to the question-mark **(?)** character. In this case, if the first character in *optstring* is a colon **(:),** the shell variable **OPTARG** will be set to the option character found, but no output will be written to standard error; otherwise, the shell variable **OPTARG** will be unset and a diagnostic message will be written to standard error. This condition is considered to be an error detected in the way arguments were presented to the invoking application, but is not 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* will be set to the colon character and the shell variable **OPTARG** will be set to the option character found.
- Otherwise, the shell variable specified by *name* will be set to the question-mark character, the shell variable **OPTARG** will be unset, and a diagnostic message will be written to standard error. This condition is considered to be an error detected in the way arguments were presented to the invoking application, but is not an error in **getopts** processing; a diagnostic message will be written as stated, but the exit status will be zero.

When the end of options is encountered, the **getopts** utility will exit with a return value greater than zero; the shell variable **OPTIND** will be set to the index of the first non-option-argument, where the first −− argument is considered to be an option-argument if there are no other non-option-arguments appearing before it, or the value **$#** + 1 if there are no non-option-arguments; the *name* variable will be set to the question-mark character. Any of the following identifies 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** are local to the caller of **getopts** and are not exported by default.

The shell variable specified by the *name* operand, **OPTIND** and **OPTARG** affect the current 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* values. 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.

sh

getopts is a built-in Bourne shell command used to parse positional parameters and to check for valid options. See **sh**(1). It supports all applicable rules of the command syntax standard (see Rules 3-10, **intro**(1)). It should be used in place of the **getopt** command.

optstring must contain the option letters the command using **getopts** will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, **getopts** places the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell script is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, **getopts** places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name* .

When the end of options is encountered, **getopts** exits with a non-zero exit status. The special option — may be used to delimit the end of the options.

By default, **getopts** parses the positional parameters. If extra arguments (*argument* . . .) are given on the **getopts** command line, **getopts** parses them instead.

/usr/lib/getoptcvt reads the shell script in *filename*, converts it to use **getopts** instead of **getopt**, and writes the results on the standard output.

So that all new commands will adhere to the command syntax standard described in **intro**(1), they should use **getopts** or **getopt** to parse positional parameters and check for options that are valid for that command.

Examples:

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
      case $c in
      a | b)     FLAG=$c;;
      o)     OARG=$OPTARG;;
      \?)    echo $USAGE
             exit 2;;
```

```
        esac
    done
    shift `expr $OPTIND − 1`
```

This code accepts any of the following as equivalent:

```
cmd −a −b −o "xxx z yy" filename
cmd −a −b −o "xxx z yy" −− filename
cmd −ab −o xxx,z,yy filename
cmd −ab −o "xxx z yy" filename
cmd −o xxx,z,yy −b −a filename
```

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.

Although the following command syntax rule (see **intro**(1)) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLES** section above, **a** and **b** are options, and the option **o** requires an option-argument. The following example violates Rule 5: options with option-arguments must not be grouped with other options:

```
example% cmd −aboxxx filename
```

The following example violates Rule 6: there must be white space after an option that takes an option-argument:

```
example% cmd −ab −oxxx filename
```

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

ksh Checks *arg* for legal options. If *arg* is omitted, the positional parameters are used. An option argument begins with a **+** or a −. An option not beginning with **+** or − or the argument −− ends the options. *optstring* contains the letters that **getopts** recognizes. If a letter is followed by a **:**, that option is expected to have an argument. The options can be separated from the argument by blanks.

getopts places the next option letter it finds inside variable *name* each time it is invoked with a **+** prepended when *arg* begins with a **+**. The index of the next *arg* is stored in **OPTIND**. The option argument, if any, gets stored in **OPTARG**.

A leading **:** in *optstring* causes **getopts** to store the letter of an invalid option in **OPTARG**, and to set *name* to **?** for an unknown option and to **:** when a required option is missing. Otherwise, **getopts** prints an error message. The exit status is non-zero when there are no more options.

For a further discussion of the Korn shell's **getopts** built-in command, see the previous discussion in the Bourne shell, **sh**, section of this manpage.

OPERANDS

The following operands are supported:

optstring A string containing the option characters recognised by the utility invoking **getopts**. If a character is followed by a colon, the option will 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** will 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 recognised if it is not supplied as a separate argument when **getopts** is invoked; see **getopt**(3C). The characters question-mark and colon must not be used as option characters by an application. The use of other option characters that are not alphanumeric produces unspecified results. If the option-argument is not supplied as a separate argument from the option character, the value in **OPTARG** will be stripped of the option character and the –. The first character in *optstring* will determine how **getopts** will behave if an option character is not known or an option-argument is missing.

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

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

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 will 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. 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.

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" "$*"
```

ENVIRONMENT

See **environ**(5) for descriptions of the following environment variables that affect the execution of **getopts**: **LC_CTYPE**, **LC_MESSAGES**, and **NLSPATH**.

OPTIND This variable is used by **getopts** as the index of the next argument to be processed.

EXIT STATUS

The following exit values are returned:

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

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

SEE ALSO

intro(1), **getopt**(1), **getoptcvt**(1), **ksh**(1), **sh**(1), **getopt**(3C), **environ**(5)

DIAGNOSTICS

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

- The invoking program name will be identified in the message. The invoking program name will be the value of the shell special parameter **0** 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 will be identified and the invalid option character will be identified in the message.
- If an option requiring an option-argument is found, but an option-argument is not found, this error will be identified and the invalid option character will be identified in the message.