**NAME**

getopt – get option letter from argument vector

**SYNOPSIS**

**#include <stdlib.h>**

**int getopt(int** *argc*, **char** * **const** **argv*, **const char** **optstring***);**

**extern char *optarg;**
**extern int optind, opterr, optopt;**

**MT-LEVEL**

Unsafe

**DESCRIPTION**

**getopt( )** returns the next option letter in *argv* that matches a letter in *optstring*. It supports all the rules of the command syntax standard (see **intro**(1)). Since all new commands are intended to adhere to the command syntax standard, they should use **getopts**(1), **getopt**(3C) or **getsubopt**(3C) to parse positional parameters and check for options that are legal for that command.

*optstring* must contain the option letters the command using **getopt( )** will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which may be separated from it by white space. *optarg* is set to point to the start of the option argument on return from **getopt( )**.

**getopt( )** places in *optind* the *argv* index of the next argument to be processed. *optind* is external and is initialized to 1 before the first call to **getopt( )**. When all options have been processed (that is, up to the first non-option argument), **getopt( )** returns **EOF**. The special option "−−" (two hyphens) may be used to delimit the end of the options; when it is encountered, **EOF** is returned and "−−"' is skipped. This is useful in delimiting non-option arguments that begin with ''–'' (hyphen).

**RETURN VALUES**

**getopt( )** prints an error message on the standard error and returns a "**?**" (question mark) when it encounters an option letter not included in *optstring* or no argument after an option that expects one. This error message may be disabled by setting **opterr** to 0. The value of the character that caused the error is in **optopt**.

**EXAMPLES**

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options **a** and **b**, and the option **o**, which requires an argument:

```
#include <stdlib.h>
#include <stdio.h>
main (int argc, char **argv)
{
        int c;
        extern char *optarg;
        extern int optind;
        int aflg = 0;
        int bflg = 0;
        int errflg = 0;
```

```
        char *ofile = NULL;

        while ((c = getopt(argc, argv, "abo:")) != EOF)
                switch (c) {
                case 'a':
                        if (bflg)
                                errflg++;
                        else
                                aflg++;
                        break;
                case 'b':
                        if (aflg)
                                errflg++;
                        else
                                bflg++;
                        break;
                case 'o':
                        ofile = optarg;
                        (void)printf("ofile = %s\n", ofile);
                        break;
                case '?':
                        errflg++;
                }
        if (errflg) {
                (void)fprintf(stderr,
                        "usage: cmd [-a|-b] [-o <filename>] files...\n");
                exit (2);
        }
        for ( ; optind < argc; optind++)
                (void)printf("%s\n", argv[optind]);
        return 0;
}
```

**SEE ALSO**

**intro**(1), **getopts**(1), **getopt**(3C), **getsubopt**(3C), **setlocale**(3C), **gettext**(3I)

**NOTES**

If the application is linked with −**lintl**, then messages printed from this function are in the native language specified by the **LC_MESSAGES** locale category; see **setlocale**(3C).

The library routine **getopt()** does not fully check for mandatory arguments. That is, given an option string **a:b** and the input −**a** −**b**, **getopt()** assumes that −**b** is the mandatory argument to the −**a** option and not that −**a** is missing a mandatory argument.

It is a violation of the command syntax standard (see **intro**(1)) for options with arguments to be grouped with other options, as in **cmd** −**abo** *filename* , where **a** and **b** are options, **o** is an option that requires an argument, and *filename* is the argument to **o**. Although this syntax is permitted in the current implementation, it should not be used because it may not be supported in future releases. The correct syntax to use is:

**cmd** −**ab** −**o** *filename.*