

**NAME**

getopt, getopt\_long, getopt\_long\_only, optarg, optind, opterr, optopt - Parse command-line options

**SYNOPSIS**

```
#include <unistd.h>

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

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

#include <getopt.h>

int getopt_long(int argc, char * const argv[],
                const char *optstring,
                const struct option *longopts, int *longindex);

int getopt_long_only(int argc, char * const argv[],
                     const char *optstring,
                     const struct option *longopts, int *longindex);
```

Feature Test Macro Requirements for glibc (see [feature\\_test\\_macros\(7\)](#)):

```
getopt(): _POSIX_C_SOURCE >= 2 || _XOPEN_SOURCE
getopt_long(), getopt_long_only(): _GNU_SOURCE
```

**DESCRIPTION**

The **getopt()** function parses the command-line arguments. Its arguments *argc* and *argv* are the argument count and array as passed to the *main()* function on program invocation. An element of *argv* that starts with '-' (and is not exactly "-" or "--") is an option element. The characters of this element (aside from the initial '-') are option characters. If **getopt()** is called repeatedly, it returns successively each of the option characters from each of the option elements.

The variable *optind* is the index of the next element to be processed in *argv*. The system initializes this value to 1. The caller can reset it to 1 to restart scanning of the same *argv*, or when scanning a new argument vector.

If **getopt()** finds another option character, it returns that character, updating the external variable *optind* and a static variable *nextchar* so that the next call to **getopt()** can resume the scan with the following option character or *argv*-element.

If there are no more option characters, **getopt()** returns -1. Then *optind* is the index in *argv* of the first *argv*-element that is not an option.

*optstring* is a string containing the legitimate option characters. If such a character is followed by a colon, the option requires an argument, so **getopt()** places a pointer to the following text in the same *argv*-element, or the text of the following *argv*-element, in *optarg*. Two colons mean an option takes an optional arg; if there is text in the current *argv*-element (i.e., in the same word as the option name itself, for example, "-oarg"), then it is returned in *optarg*, otherwise *optarg* is set to zero. This is a GNU extension. If *optstring* contains **W** followed by a semicolon, then **-W foo** is treated as the long option **--foo**. (The **W** option is reserved by POSIX.2 for implementation extensions.) This behavior is a GNU extension, not available with libraries before glibc 2.

By default, **getopt()** permutes the contents of *argv* as it scans, so that eventually all the nonoptions are at the end. Two other modes are also implemented. If the first character of *optstring* is '+' or the environment variable **POSIXLY\_CORRECT** is set, then option processing stops as soon as a nonoption argument is encountered. If the first character of *optstring* is '-', then each nonoption *argv*-element is handled as if it were the argument of an option with character code 1. (This is used by programs that were written to expect options and other *argv*-elements in any order and that care about the ordering of the two.) The special argument "--" forces an end of option-scanning regardless of the scanning mode.

While processing the option list, **getopt()** can detect two kinds of errors: (1) an option character that was

not specified in *optstring* and (2) a missing option argument (i.e., an option at the end of the command line without an expected argument). Such errors are handled and reported as follows:

- \* By default, **getopt()** prints an error message on standard error, places the erroneous option character in *optopt*, and returns '?' as the function result.
- \* If the caller has set the global variable *opterr* to zero, then **getopt()** does not print an error message. The caller can determine that there was an error by testing whether the function return value is '?'. (By default, *opterr* has a nonzero value.)
- \* If the first character (following any optional '+' or '-' described above) of *optstring* is a colon (':'), then **getopt()** likewise does not print an error message. In addition, it returns ':' instead of '?' to indicate a missing option argument. This allows the caller to distinguish the two different types of errors.

#### **getopt\_long() and getopt\_long\_only()**

The **getopt\_long()** function works like **getopt()** except that it also accepts long options, started with two dashes. (If the program accepts only long options, then *optstring* should be specified as an empty string (""), not NULL.) Long option names may be abbreviated if the abbreviation is unique or is an exact match for some defined option. A long option may take a parameter, of the form **--arg=param** or **--arg param**.

*longopts* is a pointer to the first element of an array of *struct option* declared in *<getopt.h>* as

```
struct option {
    const char *name;
    int has_arg;
    int *flag;
    int val;
};
```

The meanings of the different fields are:

*name* is the name of the long option.

*has\_arg*

is: **no\_argument** (or 0) if the option does not take an argument; **required\_argument** (or 1) if the option requires an argument; or **optional\_argument** (or 2) if the option takes an optional argument.

*flag* specifies how results are returned for a long option. If *flag* is NULL, then **getopt\_long()** returns *val*. (For example, the calling program may set *val* to the equivalent short option character.) Otherwise, **getopt\_long()** returns 0, and *flag* points to a variable which is set to *val* if the option is found, but left unchanged if the option is not found.

*val* is the value to return, or to load into the variable pointed to by *flag*.

The last element of the array has to be filled with zeros.

If *longindex* is not NULL, it points to a variable which is set to the index of the long option relative to *longopts*.

**getopt\_long\_only()** is like **getopt\_long()**, but '-' as well as "--" can indicate a long option. If an option that starts with '-' (not "--") doesn't match a long option, but does match a short option, it is parsed as a short option instead.

#### **RETURN VALUE**

If an option was successfully found, then **getopt()** returns the option character. If all command-line options have been parsed, then **getopt()** returns -1. If **getopt()** encounters an option character that was not in *optstring*, then '?' is returned. If **getopt()** encounters an option with a missing argument, then the return value depends on the first character in *optstring*: if it is ':', then ':' is returned; otherwise '?' is returned.

**getopt\_long()** and **getopt\_long\_only()** also return the option character when a short option is recognized. For a long option, they return *val* if *flag* is NULL, and 0 otherwise. Error and -1 returns are the same as for **getopt()**, plus '?' for an ambiguous match or an extraneous parameter.

**ENVIRONMENT****POSIXLY\_CORRECT**

If this is set, then option processing stops as soon as a nonoption argument is encountered.

**\_<PID>\_GNU\_nonoption\_argv\_flags\_**

This variable was used by [bash\(1\)](#) 2.0 to communicate to glibc which arguments are the results of wildcard expansion and so should not be considered as options. This behavior was removed in [bash\(1\)](#) version 2.01, but the support remains in glibc.

**ATTRIBUTES**

For an explanation of the terms used in this section, see [attributes\(7\)](#).

Interface	Attribute	Value
<code>getopt()</code> , <code>getopt_long()</code> , <code>getopt_long_only()</code>	Thread safety	MT-Unsafe race:getopt env

**CONFORMING TO****getopt():**

POSIX.1-2001, POSIX.1-2008, and POSIX.2, provided the environment variable **POSIXLY\_CORRECT** is set. Otherwise, the elements of *argv* aren't really *const*, because we permute them. We pretend they're *const* in the prototype to be compatible with other systems.

The use of '+' and '-' in *optstring* is a GNU extension.

On some older implementations, `getopt()` was declared in `<stdio.h>`. SUSv1 permitted the declaration to appear in either `<unistd.h>` or `<stdio.h>`. POSIX.1-1996 marked the use of `<stdio.h>` for this purpose as LEGACY. POSIX.1-2001 does not require the declaration to appear in `<stdio.h>`.

**getopt\_long() and getopt\_long\_only():**

These functions are GNU extensions.

**NOTES**

A program that scans multiple argument vectors, or rescans the same vector more than once, and wants to make use of GNU extensions such as '+' and '-' at the start of *optstring*, or changes the value of **POSIXLY\_CORRECT** between scans, must reinitialize `getopt()` by resetting *optind* to 0, rather than the traditional value of 1. (Resetting to 0 forces the invocation of an internal initialization routine that rechecks **POSIXLY\_CORRECT** and checks for GNU extensions in *optstring*.)

**EXAMPLE****getopt()**

The following trivial example program uses `getopt()` to handle two program options: *-n*, with no associated value; and *-t val*, which expects an associated value.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int
main(int argc, char *argv[])
{
    int flags, opt;
    int nsecs, tfnd;

    nsecs = 0;
    tfnd = 0;
    flags = 0;
    while ((opt = getopt(argc, argv, "nt:")) != -1) {
        switch (opt) {
            case 'n':
                flags = 1;

```

```

break;
case 't':
nsecs = atoi(optarg);
tfnd = 1;
break;
default: /* '?' */
fprintf(stderr, "Usage: %s [-t nsecs] [-n] name\n",
argv[0]);
exit(EXIT_FAILURE);
}
}

printf("flags=%d; tfnd=%d; nsecs=%d; optind=%d\n",
flags, tfnd, nsecs, optind);

if (optind >= argc) {
fprintf(stderr, "Expected argument after options\n");
exit(EXIT_FAILURE);
}

printf("name argument = %s\n", argv[optind]);

/* Other code omitted */

exit(EXIT_SUCCESS);
}

```

**getopt\_long()**

The following example program illustrates the use of **getopt\_long()** with most of its features.

```

#include <stdio.h> /* for printf */
#include <stdlib.h> /* for exit */
#include <getopt.h>

int
main(int argc, char **argv)
{
int c;
int digit_optind = 0;

while (1) {
int this_option_optind = optind ? optind : 1;
int option_index = 0;
static struct option long_options[] = {
{"add", required_argument, 0, 0 },
{"append", no_argument, 0, 0 },
{"delete", required_argument, 0, 0 },
{"verbose", no_argument, 0, 0 },
{"create", required_argument, 0, 'c'},
{"file", required_argument, 0, 0 },
{0, 0, 0, 0 }
};

c = getopt_long(argc, argv, "abc:d:012",
long_options, &option_index);
if (c == -1)
break;

switch (c) {
case 0:

```

```

printf("option %s", long_options[option_index].name);
if (optarg)
printf(" with arg %s", optarg);
printf("\n");
break;

case '0':
case '1':
case '2':
if (digit_optind != 0 && digit_optind != this_option_optind)
printf("digits occur in two different argv-elements.\n");
digit_optind = this_option_optind;
printf("option %c\n", c);
break;

case 'a':
printf("option a\n");
break;

case 'b':
printf("option b\n");
break;

case 'c':
printf("option c with value '%s'\n", optarg);
break;

case 'd':
printf("option d with value '%s'\n", optarg);
break;

case '?':
break;

default:
printf("?? getopt returned character code 0%o ??\n", c);
}
}

if (optind < argc) {
printf("non-option ARGV-elements: ");
while (optind < argc)
printf("%s ", argv[optind++]);
printf("\n");
}

exit(EXIT_SUCCESS);
}

```

**SEE ALSO**

[getopt\(1\)](#), [getsubopt\(3\)](#)

**COLOPHON**

This page is part of release 4.10 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.