NAME

getopt − 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*;

#define _GNU_SOURCE
#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*);

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.

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, 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 behaviour is a GNU extension, not available with libraries before GNU libc 2.

By default, **getopt()** permutes the contents of *argv* as it scans, so that eventually all the non-options 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 non-option argument is encountered. If the first character of *optstring* is '-', then each non-option *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.

If **getopt()** does not recognize an option character, it prints an error message to stderr, stores the character in *optopt*, and returns '?'. The calling program may prevent the error message by setting *opterr* to 0.

The **getopt_long()** function works like **getopt()** except that it also accepts long options, started out by two dashes. 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 zeroes.

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

The **getopt()** function returns the option character if the option was found successfully, ':' if there was a missing parameter for one of the options, '?' for an unknown option character, or −1 for the end of the option list.

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 VARIABLES

POSIXLY_CORRECT
If this is set, then option processing stops as soon as a non-option argument is encountered.

_<PID>_GNU_nonoption_argv_flags_
This variable was used by **bash** 2.0 to communicate to GNU libc which arguments are the results of wildcard expansion and so should not be considered as options. This behaviour was removed in **bash** version 2.01, but the support remains in GNU libc.

EXAMPLE

The following example program, from the source code, illustrates the use of **getopt_long()** with most of its features.

```
#include <stdio.h>

int
main (argc, argv)
    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", 1, 0, 0},
        {"append", 0, 0, 0},
        {"delete", 1, 0, 0},
        {"verbose", 0, 0, 0},
        {"create", 1, 0, 'c'},
        {"file", 1, 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 (0);
}
```

BUGS

This manpage is confusing.

The POSIX.2 specification of **getopt()** has a technical error described in POSIX.2 Interpretation 150. The GNU implementation (and probably all other implementations) implements the correct behaviour rather than that specified.

CONFORMING TO

getopt():

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.