Package 'getopt'

May 8, 2026

Encoding UTF-8

Type Package

Title C-Like 'getopt' Behavior

Version 1.21.1

URL https://github.com/trevorld/r-getopt,
https://trevorldavis.com/R/getopt/

BugReports https://github.com/trevorld/r-getopt/issues

Description Package designed to be used with Rscript to write
'#!' shebang scripts that accept short and long flags/options.
Many users will prefer using instead the packages optparse or argparse
which add extra features like automatically generated help option and usage,
support for default values, positional argument support, etc.

License GPL (>= 2)

Depends R (>= 3.3)

Suggests stats, testthat

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

Author Trevor L. Davis [aut, cre] (ORCID:
<https://orcid.org/0000-0001-6341-4639>),
Allen Day [aut] (Original package author),
Roman Zenka [ctb]

Maintainer Trevor L. Davis <trevor.l.davis@gmail.com>

Repository CRAN

Date/Publication 2026-04-27 18:50:02 UTC

Contents

`getfile` *Returns file name being interpreted by Rscript*

Description

`getfile()` returns the file name that `Rscript` is interpreting. Also supports `littler` via the `LITTLER_SCRIPT_PATH` environment variable. `get_Rscript_filename()` is an alias.

Usage

```
getfile()

get_Rscript_filename()
```

Value

A string with the filename of the calling script. If not found (i.e. you are in a interactive session) returns `NA_character_`.

`getoperand` *Extract positional arguments from a getopt result*

Description

Extracts the positional arguments stored in the `"operands"` attribute of the object returned by `getopt()`.

Usage

```
getoperand(x)
```

Arguments

`x`	An object of class `"getopt"` as returned by `getopt()`.

Value

A character vector of positional arguments.

Examples

```
spec <- matrix(c("verbose", "v", 0, "logical"), ncol = 4, byrow = TRUE)
opt <- getopt(spec, c("--verbose", "--", "file1.txt", "file2.txt"))
getoperand(opt)
```

`getopt` *C-like getopt behavior*

Description

`getopt` is primarily intended to be used with `Rscript`. It facilitates writing `#!` shebang scripts that accept short and long flags/options. It can also be used from `R` directly, but is probably less useful in this context.

Usage

```
getopt(
  spec = NULL,
  opt = NULL,
  command = getfile(),
  usage = FALSE,
  debug = FALSE,
  operand = "after--only"
)
```

Arguments

`spec` The getopt specification, or spec of what options are considered valid. The specification must be either a 4-5 column matrix, a 4-5 column data frame, or a character vector coercible into a 4 column matrix using `matrix(x, ncol = 4L, byrow = TRUE)` command. The matrix/vector contains:

Column 1: the *long flag* name. A multi-character string.

Column 2: *short flag* alias of Column 1. A single-character string. May be `NA_character_` if there is no short flag.

Column 3: *Action* of the *flag*. A string. Possible values:

- `"append"` (flag takes a required argument; appends it to a vector each time the flag is used)
- `"count"` (flag takes no argument; stores count of how many times the flag was present)
- `"store_true"` (flag takes no argument; stores `TRUE`)
- `"store_false"` (flag takes no argument; stores `FALSE`)
- `"store"` (flag takes a required argument)
- `"store_optional"` (flag takes an optional argument but if none present stores `TRUE`)

For backwards compatibility `0`, `1`, `2` are accepted as aliases for `"store_true"`, `"store"`, `"store_optional"` respectively.

Column 4: Data type to which the *flag*'s argument shall be cast using `storage.mode()`. A multi-character string. Only used when an action took an argument. Possible values: "logical", "integer", "double", "complex", "character". "numeric" is treated as an alias for "double".

Column 5 (optional): A brief description of the purpose of the option.

The terms *option*, *flag*, *long flag*, *short flag*, and *argument* have very specific meanings in the context of this document. Read the "Details" section of `getopt()` for definitions.

`opt` This defaults to the return value of `commandArgs(TRUE)` unless `argv` is in the global environment in which case it uses that instead (this is for compatibility with `littler`).

If R was invoked directly via the `R` command, this corresponds to all arguments passed to R after the `--args` flag.

If R was invoked via the `Rscript` command, this corresponds to all arguments after the name of the R script file.

Read about `commandArgs()` and Rscript to learn more.

`command` The string to use in the usage message as the name of the script. See argument *usage*.

`usage` If `TRUE`, argument `opt` will be ignored and a usage statement (character string) will be generated and returned from `spec`.

`debug` This is used internally to debug the `getopt()` function itself.

`operand` Controls how positional arguments (operands) are handled.

- `"after--only"` (the default) only collects operands that appear after a `"--"` separator.
- `"strict"` (in addition to operands that appear after a `"--"` separator) collects tokens that do not look like flags and were not consumed as a flag argument, while still erroring on unrecognized flags.

Operands are stored in the `"operand"` attribute of the result and can be retrieved with `getoperand()`.

Details

Notes on naming convention:

1. An *option* is one of the shell-split input strings.
2. A *flag* is a type of *option*. a *flag* can be defined as having no *argument* (defined below), a required *argument*, or an optional *argument*.
3. An *argument* is a type of *option*, and is the value associated with a flag.
4. A *long flag* is a type of *flag*, and begins with the string `--`. If the *long flag* has an associated *argument*, it may be delimited from the *long flag* by either a trailing `=`, or may be the subsequent *option*.
5. A *short flag* is a type of *flag*, and begins with the string `-`. If a *short flag* has an associated *argument*, it is the subsequent *option*. *short flags* may be bundled together, sharing a single leading `-`, but only the final *short flag* is able to have a corresponding *argument*.

Many users wonder whether they should use the `getopt` package, `optparse` package, or `argparse` package. Here is some of the major differences:

Features available in `getopt` unavailable in `optparse`

1. `getopt` allows one to specify options with an optional argument.
2. Long flags may be abbreviated as long as the abbreviation is unique, e.g. `--verb` matches `--verbose` if no other long flag starts with `verb`.

Some features implemented in the `optparse` package unavailable in `getopt`

1. Support for capturing positional arguments after the optional arguments when `positional_arguments` set to `TRUE` in `optparse::parse_args()`
2. Automatic generation of an help option and printing of help text when encounters an `-h`
3. Option to specify default arguments for options as well the variable name to store option values

There is also package `argparse` supports all the features of both getopt and optparse (plus more)

Some Features unlikely to be implemented in `getopt`:

1. Support for lists, e.g. `--define os=linux --define os=redhat` would set `result$os$linux=TRUE` and `result$os$redhat=TRUE`.

Value

`getopt()` returns a list data structure containing names of the flags that were present in the character vector passed in under the `opt` argument. Each value of the list is coerced to the data type specified according to the value of the `spec` argument. See the "Details" section for more information. Any positional arguments (operands) are stored in its `"operand"` attribute and can also be accessed by `getoperand()`.

Author(s)

Allen Day and Trevor L. Davis

Examples

```
#!/path/to/Rscript
library('getopt')
# get options, using the spec as defined by the matrix
spec <- matrix(c(
  'verbose', 'v', 2, "integer",
  'help'   , 'h', 0, "logical",
  'count'  , 'c', 1, "integer",
  'mean'   , 'm', 1, "double",
  'sd'     , 's', 1, "double"
), byrow = TRUE, ncol = 4L)
opt <- getopt(spec)

# if help was asked for print a friendly message and exit
if (isTRUE(opt$help)) {
  cat(getusage(spec))
```

```
  q(status = 0)
}

# set reasonable defaults for options that were not specified
if (is.null(opt$mean)) opt$mean <- 0
if (is.null(opt$sd)) opt$sd <- 1
if (is.null(opt$count)) opt$count <- 10L
if (is.null(opt$verbose)) opt$verbose <- FALSE

# print some progress messages to stderr, if requested
if (opt$verbose) write("writing...", stderr())

# do some operation based on user input
cat(rnorm(opt$count, mean = opt$mean, sd = opt$sd), sep = "\n")
```

`getusage` *Generate a usage string*

Description

Generate a usage string from a getopt spec matrix.

Usage

```
getusage(spec, command = getfile(), usage = "Usage: %command %options")
```

Arguments

`spec`	The getopt specification, or spec of what options are considered valid. The specification must be either a 4-5 column matrix, a 4-5 column data frame, or a character vector coercible into a 4 column matrix using `matrix(x, ncol = 4L, byrow = TRUE)` command. The matrix/vector contains: Column 1: the *long flag* name. A multi-character string. Column 2: *short flag* alias of Column 1. A single-character string. May be `NA_character_` if there is no short flag. Column 3: *Action* of the *flag*. A string. Possible values: • `"append"` (flag takes a required argument; appends it to a vector each time the flag is used) • `"count"` (flag takes no argument; stores count of how many times the flag was present) • `"store_true"` (flag takes no argument; stores `TRUE`) • `"store_false"` (flag takes no argument; stores `FALSE`) • `"store"` (flag takes a required argument) • `"store_optional"` (flag takes an optional argument but if none present stores `TRUE`)

For backwards compatibility `0`, `1`, `2` are accepted as aliases for `"store_true"`, `"store"`, `"store_optional"` respectively.

Column 4: Data type to which the *flag*'s argument shall be cast using `storage.mode()`. A multi-character string. Only used when an action took an argument. Possible values: "logical", "integer", "double", "complex", "character". "numeric" is treated as an alias for "double".

Column 5 (optional): A brief description of the purpose of the option.

The terms *option*, *flag*, *long flag*, *short flag*, and *argument* have very specific meanings in the context of this document. Read the "Details" section of `getopt()` for definitions.

`command` The string to use in the usage message as the name of the script. See argument *usage*.

`usage` A template string for the usage line. `"%command"` is replaced by the value of `command` and `"%options"` is replaced by the computed options string.

Value

A character string with the usage message.

Examples

```
spec <- matrix(c(
  'verbose', 'v', 2, "integer",
  'help'   , 'h', 0, "logical",
  'count'  , 'c', 1, "integer",
  'mean'   , 'm', 1, "double",
  'sd'     , 's', 1, "double"
), byrow = TRUE, ncol = 4)
cat(getusage(spec, command = "myscript"))
cat(getusage(spec, command = "myscript",
             usage = "Usage: %command %options FILE"))
```

---

sort_list *Recursively sorts a list*

Description

`sort_list()` returns a recursively sorted list

Usage

```
sort_list(unsorted_list)
```

Arguments

`unsorted_list` A list.

Value

A sorted list.

Examples

```
l <- list(b = 2, a = 1)
sort_list(l)
```

Index