Package 'arrg'

September 16, 2026

Version 0.2.0

Date 2026-09-16

Title Flexible Argument Parsing for R Scripts

Imports ore

Suggests tinytest, covr

Description Argument parsing for R scripts, with support for long and short Unix-style options including option clustering, positional arguments including those of variable length, and multiple usage patterns which may take different subsets of options.

Encoding UTF-8

License GPL-2

URL https://github.com/jonclayden/arrg

BugReports https://github.com/jonclayden/arrg/issues

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Jon Clayden [cre, aut] (ORCID: <https://orcid.org/0000-0002-6608-0619>)

Maintainer Jon Clayden <code@clayden.org>

Repository CRAN

Date/Publication 2026-09-16 21:40:10 UTC

Contents

arrg *Create an argument parser*

Description

This function creates an argument parser that handles the specified options and usage patterns. To parse arguments or display usage information, the methods `parse` or `show` contained in the return value should be called.

Usage

```
arrg(name, ..., patterns = list(), help = TRUE, header = NULL, footer = NULL)

## S3 method for class 'arrgParser'
print(x, ...)
```

Arguments

`name`	The name of the command.
`...`	Option specifications (see `opt()` for details). For the `print` method, further arguments to the parser's `show` method, notably `width`.
`patterns`	A list of usage patterns that are valid for the command, each specifying acceptable options and positional arguments, or a single such pattern. See `pat()` for details. If none is given, one is generated that accepts all of the command's options and any number of positional arguments, which are named `args`.
`help`	Whether to provide a help option, and a usage pattern for it, if the command does not specify one of its own. A string may be given instead of `TRUE`, and is used as the option's description. A generated option of this kind is listed first, and takes no part in a pattern's `.options=TRUE`, nor in the function returned by the `run` method.
`header`, `footer`	Optional paragraphs of text to be prepended and/or appended to the usage text produced by the `show` method of the return value. Typically used to introduce the command or give brief guidance on usage.
`x`	An argument parser, of class `"arrgParser"`.

Details

Options are handled in long form, as in `--times=3` or `--times 3`, or in short form, as in `-n3` or `-n 3`. Several short-form options may be clustered behind a single hyphen, with any option that takes an argument coming last, as in `-tn3`. Options and positional arguments may be freely interleaved. A `--` argument stops option parsing and is discarded: everything after it is treated as positional, even if it begins with a hyphen. A lone `-` is always positional, by convention referring to standard input.

The `run` method provides a script's entry point. Given a function holding the body of the script, it either calls it or returns a function that will, according to how the script was invoked. Run from a command line, by `Rscript` or `littler`, the arguments are parsed, a request for help is answered

with the usage summary, a usage error is reported on standard error with a non-zero exit status, and otherwise the body is called. When the script is `source()`d instead, nothing is run: the value is a function whose formal arguments correspond to the parser's options and positional arguments, so that the same script can be driven interactively.

The body may be a function of one argument, in which case it receives the parsed arguments as a list, conventionally called `opts`, or none, in which case they are bound in the environment it runs in and may be referred to by name. Note that such bindings mask anything of the same name in the enclosing scope, and that a name the parser could have produced but didn't is `NULL`. A block of code in braces may also be given in place of a function, and is equivalent to a function of no arguments. The braces are required: any other expression is evaluated, and must produce a function, which allows a body to be built by a factory or taken from a variable.

Value

A list of class `"arrgParser"`, with elements

- `name`: The name of the command, as given above.
- `parse(argv)`: A method to parse the character vector of arguments passed in, or by default, the value of `commandArgs(trailingOnly=TRUE)`.
- `show(con, width)`: A method to print a usage summary, detailing the valid options and patterns. Text will be printed to the specified connection, default `stdout()`, and wrapped to the width given, which defaults to the value of the standard `width` option. Any default value for an option's argument is appended to that option's description.
- `run(body, argv, execute, help, exit)`: A method designed to wrap the body of a script, given as a function or a block of code in braces, or return a function that will. `argv` overrides the arguments to parse, `execute` forces the choice between running the body (`TRUE`) and returning a function (`FALSE`), `help` names the option that requests usage information, and `exit` controls whether the R session is ended after help is given or a usage error reported. See Details.

Note

The option and pattern specifications given to this function are evaluated with `opt()` and `pat()` in scope, so a script may call `arrg::arrg()` without attaching the package's namespace, and without namespacing each of those nested calls.

Author(s)

Jon Clayden

See Also

`opt()`, `pat()`

Examples

```
# A simple parser for a command called "greet" with only one option, -n
greet <- arrg("greet", opt("n,name", "Who to greet", default="world"),
              patterns=list(pat(.options="n")))
```

```
# The body of a script. When the script is called from a command line
# run() calls this directly; when it is source()d, run() instead returns
# a function, as forced here. The mode is detected automatically by default
hello <- greet$run(function () cat("Hello,", name, "\n"), execute=FALSE)

hello()
hello(name="reader")

print(greet)
```

---

opt *Specify an option in long or short form*

Description

This function specifies an option that is accepted by an argument parser. The results of one or more calls to this function are typically passed to `arrg()`.

Usage

```
opt(label, description, arg = FALSE, default = NULL)
```

Arguments

`label`	A short-form (single character) and/or long-form label for the option, specified comma-separated in a single string. At most one of each form must be given. Long-form labels may be internally hyphenated, as in "dry-run". Leading hyphens and surrounding whitespace are optional, and will be stripped.
`description`	A textual description of the option, for use in the usage summary.
`arg`	The name of the option's argument, if it takes one. Otherwise `FALSE`, indicating no argument. If a `default` is given then the option takes an argument whatever the value of this parameter, and the argument will be named after the option unless a name is given here.
`default`	A default value for the argument, if one is accepted. This does not have to be a string, and arguments will be coerced to match the mode of the default when parsed. The default value of `NULL` means that no default is specified: an option taking an argument will then default to `NA`, and an option taking no argument to `FALSE`.

Value

A list of class `"arrgOption"` giving details of the option. This will not usually be used directly, but passed to `arrg()`.

Author(s)

Jon Clayden

See Also

`arrg()`

Examples

```
# A simple flag-style option with no argument
opt("h,help", "Display this usage information and exit")

# An option that takes an integer argument called "count"
opt("n,times", "Run this many times", arg="count", default=1L)
```

`pat` *Specify a usage pattern*

Description

This function is used to specify a valid usage pattern for the command, which may be one of a number of mutually exclusive patterns available. Its return value is generally passed to `arrg()`.

Usage

```
pat(..., .options = NULL)
```

Arguments

`...`	Character strings naming positional arguments, if any are valid. Positional arguments are required by default; if not required they should be followed by a question mark. Optional arguments must come after all required ones. The final positional argument (only) may take multiple values, in which case it should contain an ellipsis (...), before the question mark if the argument is also optional. An argument may instead be given as a named element, as in `pat(path=".")`, in which case the name is the specification and the value is a default. Such an argument is optional, and a value given for it will be coerced to the mode of the default, as for `opt()`.
`.options`	A string naming the long or short labels of options that can be specified with this pattern, comma-separated. Short form options may be given in one letter cluster for convenience. Options are only required if followed by an exclamation mark. Alternatively `TRUE`, meaning every option that the command declares, all of them optional, except any that `arrg()` generated itself, such as an automatic help option. The leading period distinguishes this parameter from the positional arguments passed in `...`, whose names can never contain one.

Details

When parsing arguments, patterns are tried in the order specified, and the first valid pattern will be chosen. A pattern will be considered a valid match if all required options and positional arguments have been specified, and no unexpected options are included.

Value

A list of class "arrgPatternSpec", capturing the positional arguments, with options in an attribute. This will not usually be used directly, but passed to arrg().

Author(s)

Jon Clayden

See Also

arrg()

Examples

```
# A pattern with no positional arguments, but requiring the -h flag
pat(.options="h!")

# A pattern that takes a command and variable number of arguments, and
# accepts the -n and -t options (note the latter are specified in cluster
# form, but "n,t" is also valid)
pat("command", "arg...?", .options="nt")

# A pattern with one optional argument, which defaults to "." if it is
# not given, and which accepts every option the command declares
pat(path=".", .options=TRUE)
```

Index