Package ‘optigrab’

January 7, 2019

Type Package
Title Command-Line Parsing for an R World
Version 0.9.2.1
Date 2019-01-05
Maintainer Christopher Brown <chris.brown@decisionpatterns.com>

URL https://github.com/decisionpatterns/optigrab
BugReports https://github.com/decisionpatterns/optigrab/issues

Description Parse options from the command-line using a simple, clean syntax.
It requires little or no specification and supports short and long options,
GNU-, Java- or Microsoft- style syntaxes, verb commands and more.

Depends R (>= 3.0.0), methods
Imports stringi (>= 0.4.1), magrittr (>= 1.5)
Suggests testthat (>= 0.9.1), knitr (>= 1.9), rmarkdown
License GPL (>= 3) | file LICENSE
LazyLoad no
VignetteBuilder knitr

RoxygenNote 6.1.1
Encoding UTF-8
Repository CRAN

NeedsCompilation no

Author Christopher Brown [aut, cre],
Daryl McCullough [aut],
Decision Patterns [cph]

Date/Publication 2019-01-07 18:40:03 UTC
R topics documented:

- gnu_style
- is_flag
- java_style
- ms_style
- optigrab
- opt_assign
- opt_expand
- opt_fill
- opt_get
- opt_get_args
- opt_get_path
- opt_get_verb
- opt_help
- opt_split_args
- opt_style
- str_to_opts
- this_file

Index

<table>
<thead>
<tr>
<th>gnu_style</th>
<th>GNU-style command line options</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>treatments</td>
<td>3</td>
</tr>
<tr>
<td>java_style</td>
<td>4</td>
</tr>
<tr>
<td>ms_style</td>
<td>4</td>
</tr>
<tr>
<td>optigrab</td>
<td>5</td>
</tr>
<tr>
<td>opt_assign</td>
<td>6</td>
</tr>
<tr>
<td>opt_expand</td>
<td>7</td>
</tr>
<tr>
<td>opt_fill</td>
<td>8</td>
</tr>
<tr>
<td>opt_get</td>
<td>9</td>
</tr>
<tr>
<td>opt_get_args</td>
<td>11</td>
</tr>
<tr>
<td>opt_get_path</td>
<td>12</td>
</tr>
<tr>
<td>opt_get_verb</td>
<td>12</td>
</tr>
<tr>
<td>opt_help</td>
<td>13</td>
</tr>
<tr>
<td>opt_split_args</td>
<td>14</td>
</tr>
<tr>
<td>opt_style</td>
<td>15</td>
</tr>
<tr>
<td>str_to_opts</td>
<td>16</td>
</tr>
<tr>
<td>this_file</td>
<td>17</td>
</tr>
</tbody>
</table>

Index

Description

Functions for enabling GNU-style command-line option behavior.

Usage

gnu_style

Format

An object of class list of length 3.

Details

Functions for enabling GNU-style command-line options. GNU-style options are characterized by a single dash (−) before single character option flags and a double dash (−−) before multiple character option flags.

By convention, gnu style options flags must begin with a letter; if numbers were allowed option flags would be ambiguous with negative option values.

References

is_flag

Determine if/which vector element are options flags

Description

Determines if an element of a vector is an option flag (as opposed to a value) by checking against the option style.

Usage

is_flag(x)
which.flag(x)

Arguments

x vector of options, for example commandArgs().

Details

is_flag and which.flag are internal functions not expected to be called directly.
They are used to identify which elements of the option vector are option names (as opposed to option values). Options are identified by getOptions('optigrab')$style$flag_test. By default, optigrab follows GNU-style command line arguments, i.e. those beginning with "--" or "-" and are set at the time of package loading.

Value

logical. indicating which arguments are flags.
numeric

Examples

optigrab:::is_flag( c( "--foo", "bar") )
optigrab:::is_flag( c( "--foo", "bar", "-f", "-b", "text" ) )
optigrab:::which.flag( c( "--foo", "bar") )
optigrab:::which.flag( c( "--foo", "bar", "-f", "-b", "text" ) )
### java_style

**Java-style command line options**

**Description**

Functions for enabling Java-style command-line option behavior.

**Usage**

```r
java_style
```

**Format**

An object of class `list` of length 3.

**Details**

Functions for enabling Java-style command-line options. Java-style options are characterized by a single dash (`-`) before the option name.

By conventions, Java-style options cannot begin with a lower or upper case letter.

**See Also**

- Non-exported function `*_flag_test`, `*_flag_to_name` and `*_name_to_flag`
- `gnu_style`
- `java_style`
- `ms_style`

### ms_style

**Microsoft-style command line options**

**Description**

Functions for enabling Microsoft-style command-line option behavior.

**Usage**

```r
ms_style
```

**Format**

An object of class `list` of length 3.
Details

Functions for enabling Microsoft-style command-line options. Microsoft-style options are characterized by a single forward slash (/) before the option name.

Microsoft-style options can be supported by setting

See Also

- Non-exported function *flag_test, *flag_to_name and *name_to_flag
- gnu_style
- java_style
- ms_style

Examples

```r
opt_style(ms_style)
opt_get("foo", opts=c("/foo", "bar"))
```

---

**Description**

The optigrab package provides a function opt_grab to retrieve options/arguments from the command line. It is useful for running R in batch mode with `CMD BATCH ...` or `Rscript`.

GNU-, Java- and Microsoft-style command line options are supported. GNU-style is the default. See the `optigrab` vignettes or the github README file for details.

See Also

- `opt_get()`
- `base::commandArgs()`

Examples

```r
## Not run:
opt_get( c("foo","f"))

## End(Not run)

opts <- c("--flag", "bar")
flag <- opt_get( c("foo","f"), opts=opts ) # bar
```
opt_assign

Parse options and assign values

Description

Combines opt_get and assign for convenience.

Usage

opt_assign(x, pos = 1, inherits = FALSE, name = x, ..., assign.na = FALSE)
opt_assign_all(x, ..., assign.na = FALSE)

Arguments

x character or list; variable names or alias. If no coercion is done, and the first
element of a character vector of length greater than one will be used, with a
warning.
pos where to do the assignment. By default, assigns into the current environment.
See 'Details' for other possibilities.
inherits logical; should the enclosing frames of the environment be inspected? This is
argument is supplied to base::assign().
name character; name(s) of option as passed to opt_get. (Default: ‘x’)
... arguments passed to opt_get
assign.na logical; whether NA can be assigned. (Default: FALSE)

opt_assign combines opt_get and assign in one call. The name argument is
passed as opt_get(name=...); it defaults to x so it is generally not needed.
opt_assign_all does multiple assignments but does not allow for a names arg-
ments. Values are names by x.

See Also

* opt_get()
* base::assign()

Examples

opt_assign( "foo", opts=c("--foo","bar") )
opt_assign( c("foo","f"), opts=c("--foo","baz") )
**opt_expand**

*Expand opts to full vector*

---

**Description**

Expand option vector to split names from values. This is an internal function and should generally not be called directly.

**Usage**

```r
opt_expand(opts = commandArgs())
```

**Arguments**

- `opts` character vector of arguments. (Default: `base::commandArgs()`)

**Details**

`opt_expand` does two things:

1. Removes values preceding and including `--args`.
2. Splits and value containing an equal (`=`) sign.

**Note**

non-exported

**See Also**

- `opt_grab()`
- `base::commandArgs()`

**Examples**

```r
optigrab::opt_expand()
optigrab::opt_expand( opts=c("--foo", "bar") )
```
Description

Fill a recursive structure with command-line arguments

Usage

\[
\text{opt\_fill}(x, \text{opts} = \text{commandArgs}(), \text{style} = \text{getOption}(\text{"optigrab"})$\text{style})
\]

Arguments

- \(x\): list-like (recursive) object with names to use as a template.
- \(\text{opts}\): character command-line option list (Default: \text{commandArgs}() )
- \(\text{style}\): string; the command-line style (Default: \text{getOption}(\text{"optigrab"})$\text{style}

Details

\text{opt\_fill} uses \(x\) as a template of values to be retrieved. Named elements of \(x\) are retrieved from the command line using \text{opt\_get}(). Values are coerced to the type/class of the elements of \(x\).

This gives a handy way of defining and retrieving all setting at once overridding the defaults.

\text{opt\_fill} is similar to \text{utils::modifyList()} but does not work recursively.

Value

(A copy of) \(x\), with values filled from the command-line. If \(x\) is a reference structure, this is done by reference, returning the object invisibly.

See Also

\text{opt\_get()} \text{utils::modifyList()}

Examples

\begin{verbatim}
defaults <- list( foo="a", bar=1 )

opt_fill( defaults, opts=c( '--foo', 'command-line-foo' ) )
opt_fill( defaults, opts=c( '--foo', 'command-line-foo', '--bar', '9999' ) )
defaults <- as.environment(defaults)
opt_fill( defaults, opts=c( '--foo', 'env-fill', '--bar', '555' ) )

str( as.list(defaults) )
\end{verbatim}
**opt_get**

Get option's values from the command-line

**Description**

Returns value(s) from the command-line associated with the desired option.

**Usage**

```r
opt_get(name, default, n, required = FALSE, description = NULL,
        opts = commandArgs(), style = getOption("optigrab")$style)
```

`opt_get_gnu(flag, ...)```

`opt_get_ms(flag, ...)```

`opt_get_java(flag, ...)```

`opt_grab(flag, n = 1, opts = commandArgs())```

**Arguments**

- `name` character; vector of possible synonyms for the "flag" that identifies the option.
- `default` any; the value(s) provided if the flag is not found (default: NA)
- `n` integer; number of values to retrieve. See Details. (default: determined by default, see Details below.)
- `required` logical; whether the value is required. If not found or an incorrect, number of values are found, an error is thrown. (default: FALSE)
- `description` (character) message to be printed with opt_help
- `opts` character; vector to parse for options (default: commandArgs())
- `style` list; list of functions that define the parsing style
- `flag` character; vector of possible synonyms for the "flag" that identifies the option. These should be given as they would exactly appear on the command line.
- `...` additional arguments passed to opt_grab

**Details**

These functions support parsing of command arguments work when using Rscript, a #! on linux systems or R CMD BATCH. By default, they closely follows the ubiquitous GNU standards for command-line interfaces. `opt_grab` is the workhorse that does the actual parsing. It returns the options values or NA if it cannot discern them. It is currently exported, but this may change in future version to be an internal function. Its interface is not guarantted. The user should use `opt_get` instead.

`opt_get` supports default values, automated guessing for n and (attempts a) coercion of the return values to the correct class.
opt_get

Value

opt_grab always returns a character; it is either the value for the flags or NA_character_ if if they cannot be parsed.

opt_get returns a value the command-line value as specified by the arguments or produces an error if the value could not be determined and required==TRUE.

Selecting n

Except in rare-cases, the user should not have to specify n. This is determined from the value of default.

If default is a logical value, i.e. TRUE or FALSE, n is assumed to take no arguments. Presents of the flag on the command line will return TRUE, absense of the flag returns false.

If default is another type than logical, n is selected as length(default).

If default is missing, n is 1.

automatic coercion

Command-line arguments are character vectors. If default is supplied then the opt_get attempts to coerce the values it returns to class(default). The user might wish to supply the correct methods to handle the conversions.

side-effects

opt_grab has the additional side-effect of keeping track of the arguments. This is useful for keeping track of
flag is used to identify the command line flag. It can include all synonyms for the flags.

n the number of value(s) to retrieve from the command line. If n=0, then a logical value is returned indicating whether the flag exists

required indicates if a value is required. If the the flag is not found and there is no default given or if there is not the correct number of value(s) an error is raised.

opts is the vector from which options are parsed. By default, this is commandArgs().

References

GNU Command Line Standards

See Also

commandArgs()

Examples

```r
opts <- c( '--foo', 'bar' )

opt_get('foo')
opt_get( c('foo'), opts=opts )
opt_grab( c('--foo'), opts=opts )
```
opt_get_args

opt_get_ms(c('foo'), opts=c('/foo', 'bar'))

opt_get_java(c('foo'), opts=c('-foo', 'bar'))

# Using pipes:
## Not run:
c('foo', 'f') %>% opt_get('bar')

## End(Not run)

**opt_get_args**  
*Return arguments to Rscript*

**Description**

Arguments to a script are those following the `--args` argument.

**Usage**

```r
opt_get_args(opts = commandArgs())
```

**Arguments**

- **opts**: character; vector of arguments. (Default: `commandArgs()`)  
  Returns the user provided arguments, i.e. those following (the first) `--args` flag.  
  This is identical to what is done by `commandArgs( trailingOnly = TRUE )`  
  does. This is included an used since it supports testing/modifying the `commandArgs` array.

**Value**

character; vector stripping elements preceding and including (the first) `--args` flag.

**See Also**

- `base::commandArgs()`
- `opt_grab()`

**Examples**

```r
opt_get_args()
```

```r
opt_get_args( opts=c( "Rscript", "-a", "-b", "--args", "-c", 3, "-d" ) ) # "c" "3" "-d"
```

```r
opt_get_args( opts=c( "a", "b", "--args", "-c", "--args", "-d" ) ) # "c" "-d"
```

```r
opt_get_args( opts=c( "--foo", "bar" ) )
```
opt_get_path  

Description

Get path current running script

Usage

opt_get_path(opts = commandArgs(), full.name = FALSE)

Arguments

opts character; vector from which to parse options (default: commandArgs())
full.name boolean; expand to full path(?)

This function is deprecated, use this_file instead.

Value

character; path to Rscript or NA if there isn’t one.

References

http://stackoverflow.com/questions/1815606/rscript-determine-path-of-the-executing-script

See Also

• opt_grab()
• base::commandArgs()

Examples

optigrab::opt_get_path()

opt_get_verb  

Description

Return the verb for the application. The verb is the first argument that is not part of an option.

Usage

opt_get_verb(opts = commandArgs())
Arguments

opts character; Vector from which to parse options (default: commandArgs() )

Details

Some applications such as git support command verbs, e.g. push, fetch, etc. These style arguments can be retrieved by opt_get_verb.

opt_get_verb look for the first unaccounted for options (after --args). The number of options needed by each flags is determined by and saved by calls to opt_get(). See the details to see how n is automatically determined. When not explicitly defined the number of options needed by each flag is 1. Becasue of this, it convention to call opt_get_verb after all opt_get calls. For most simple applications, it likely doesn’t matter.

Value

character of length 1; the verb found from the command-line. NA if a verb cannot be identified.

Assumptions

opt_get_verb assumes any flags occurring before the verb have exactly 1 value. A command line such as "> myscript --verbose verb" will be misparsed; the code will assume that "verb" is the value of the flag "--verbose"

See Also

• opt_get()

• base::commandArgs()

Examples

opt_get_verb() # commandArgs()

opt_help

Automatic usage/help information

Description

Prints program usage information and, if used non-interactively, exits.

Usage

opt_help(name = c("help", "?"), opts = commandArgs())
**Arguments**

- **name** character; vector of names/aliases to use
- **opts** character; vector from which to parse options (default: `commandArgs()`)

Usage information is generated from `opt_get` calls made prior to `opt_help`. `opt_help` shused will will not be shown. It is considered best practice to handle all option parsing in a block at the beginning of the application. `opt_help()` would be best placed at the end of that block.

**Value**

logical; returns TRUE if command-line options contain a help flag, FALSE otherwise. Mainly `opt_help` is used for side-effects of printing usage/help information.

**See Also**

- `base::commandArgs()`

**Examples**

```r
opts <- c("--foo", "bar")
optigrab:::opt_grab("--foo")
optigrab:::opt_help()
```

---

**Description**

Splits command argument vector to name, value pairs. This is an internal function and should generally not be called directly.

**Usage**

```r
opt_split_args(opts = commandArgs())
```

**Arguments**

- **opts** character; vector of arguments. (Default: `commandArgs()`)

**Note**

non-exported

**See Also**

- `opt_grab()`
opt_style

Examples

    optigrab::opt_split_args()
    optigrab::opt_split_args( opts=c("--foo=hello", "-b=goodbye") )

Description

Get or sets the optigrab style

Usage

    opt_style(style)

Arguments

style  

    named list; containing the following functions: flag_test, flag_to_name and name_to_flag
    If style is not specified, opt_style gets the current optigrab style. If style is provided, it must be a named list containing three functions flag_test, flag_to_name and name_to_flag.

Value

    If style is not provided, returns a list of styles, otherwise used for the side-effect of setting the option

flag_test

    Accepts a character vector and returns a logical vector indicating whether the elements are flags.

flag_to_name

    Accepts a character vector of flags and turns them into variable names, usually by stripping delimiters that indicate that they are flags

name_to_flag

    Accepts a character vector of names and transforms them into the flags that would appear on the command line. This is used by opt_grab().

See Also

    gnu_style, java_style or ms_style
Examples

```r
opt_style()
opt_style(optigrab::gnu_style)
opt_style(optigrab::java_style)
opt_style(optigrab::ms_style)
```

---

**str_to_opts**

*Split a string bases on whitespace*

---

**Description**

Split a string based on whitespace ignore single- and double quoted entries

**Usage**

```r
str_to_opts(x = character())
```

**Arguments**

- `x` character; string to parse as if it is a command line

This is an internal function used predominantly for testing. It might be deprecated in the near future.

**Value**

A character array that could be similar to that provided by `commandArgs`.

**Note**

not-exported, by design

**See Also**

`base::commandArgs()`

**Examples**

```r
## Not run:
str <- 'cmd -t "Say Anything" --character \"Lloyd Dobler\"
str_to_opts(str)
split_ws_nonquote(str)

## End(Not run)
```
**Description**

Name or path to the current file

**Usage**

```r
this_file(opts = commandArgs(), local = TRUE, full.path = TRUE)
```

**Arguments**

- `opts`: character; vector of arguments. (Default: `commandArgs()`)
- `local`: logical; if TRUE returns the most currently sourced script as opposed to the original/first source script. (Default: TRUE)
- `full.path`: logical; Whether to return the full path to the sourced file. (Default: TRUE)

**Details**

`this_file` returns the name or path of the executing file whether the file was invoked from `Rscript` or in an interactive session. Further it source

Argument `local` controls whether it is the current file (TRUE) or the original, top-level file.

**Value**

one-element character vector with the path to the current file; returns NA in an interactive session not in a file.

**References**


**See Also**

- `opt_grab()`

**Examples**

```r
this_file()
```
Index

*Topic **datasets**
gnu_style, 2
java_style, 4
ms_style, 4

*Topic **package**
optigrab, 5

base::assignH), 6
base::commandArgsH), 5, 7, 11–14, 16
commandArgsH), 10
gnu_style, 2, 3–5, 15
is_flag, 3
java_style, 3, 4, 4, 5, 15
ms_style, 3, 4, 4, 5, 15
opt_assign, 6
opt_assign_all (opt_assign), 6
opt_expand, 7
opt_fill, 8
opt_get, 9
opt_getH), 5, 6, 8, 13
opt_get_args, 11
opt_get_gnu (opt_get), 9
opt_get_java (opt_get), 9
opt_get_ms (opt_get), 9
opt_get_path, 12
opt_get_verb, 12
opt_grab (opt_get), 9
opt_grabH), 7, 11, 12, 14, 15, 17
opt_help, 13
opt_split_args, 14
opt_style, 15
optigrab, 5
optigrab-package (optigrab), 5

str_to_opts, 16