

Parsing command-line arguments by Getopt::Long

Zuguang Gu <z.gu@dkfz.de>

April 23, 2016

There are already several R packages to parse command-line arguments such as `getopt` or *Python*-style `optparse`. Here **GetoptLong** is another command-line argument parser which wraps the powerful *Perl* module `Getopt::Long` (<http://search.cpan.org/perl/doc/Getopt%3A%3ALong>), also provides some adaptation for easier use in R.

Using **GetoptLong** is simple especially for *Perl* users because the specification is almost the same as in *Perl*. The original website of `Getopt::Long` is always your best reference.

1 Workflow of the wrapping

Figure 1 shows how the R package works for parsing the command-line arguments.

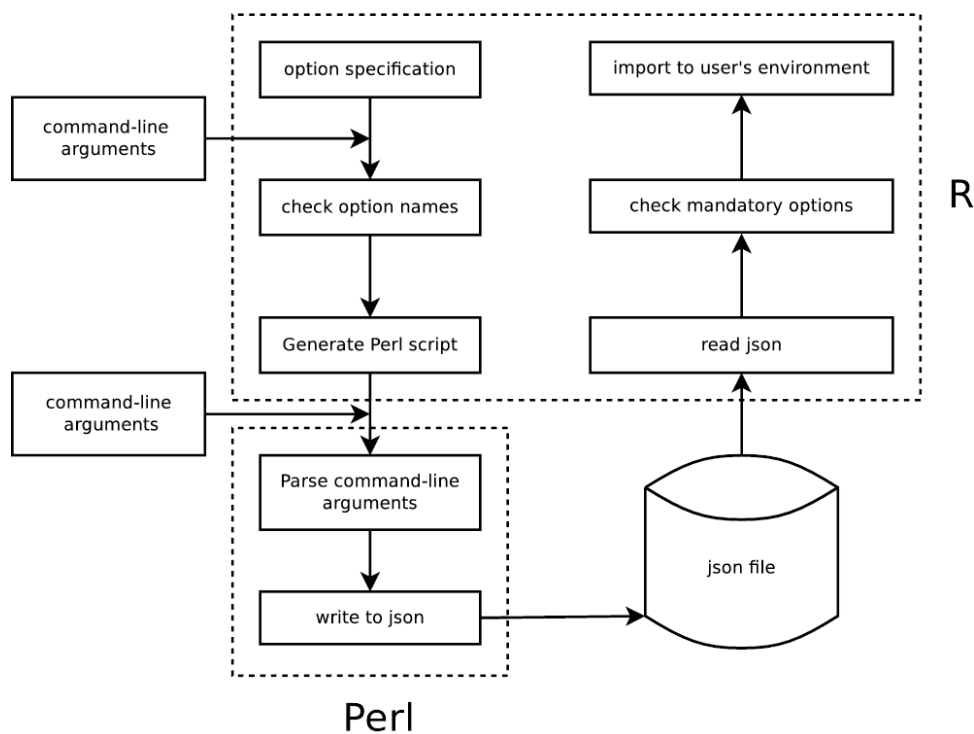


Figure 1: Workflow of wrapping

2 First example

```
library(GetoptLong)

cutoff = 0.05
GetoptLong(
```

```

"number=i", "Number of items, integer, mandatory option",
"cutoff=f", "cutoff to filter results, optional, default (0.05)",
"verbose", "print messages"
)

```

The number of arguments in `GetoptLong()` should be even number and the specification and description should always be paired.

Save the code as `test.R` and we can execute the R script as:

```

~\> Rscript test.R --number 4 --cutoff 0.01 --verbose
~\> Rscript test.R -n 4 -c 0.01 -v
~\> Rscript test.R -n 4 --verbose

```

In above example, `number` is a mandatory option and should only be integer mode. `cutoff` is optional and already has a default value. `verbose` is a logical option. If parsing is successful, two variables with name `number` and `verbose` will be imported into the working environment with specified values, and value for `cutoff` will be updated if it is specified in command-line argument.

3 Customize your options

Each specifier in options consists of two parts: the name specification and the argument specification:

```
length|size|l=i@
```

Here `length|size|l` is a list of alternative names separated by `|`. The remaining part is argument specification which defines the mode and amount of arguments. The argument specification is optional.

Specify any one of alternative option name from command-line is OK and it doesn't matter whether using one or two slash in front of the option name. Sometimes you even don't need to specify complete option names, you only need to make sure the partial name match is unique. If the partial match is not unique, it will throw an error. For above example, we can specify the argument like:

```

~\> Rscript foo.R --length 1
~\> Rscript foo.R -len 1
~\> Rscript foo.R --size 1
~\> Rscript foo.R -l 1

```

Options for argument specification are:

- no argument specification: taking no argument. Options are logical.
- `!`: taking no argument. Options are logical. You can set its opposite value by prefixing it with `no` or `no-`. E.g. `foo!` allows `--foo` as well as `--nofoo` and `--no-foo`.
- `=type[desttype][repeat]`: options should have arguments. Only either `desttype` or `repeat` can be used.

Please note “`: [type] [desttype]`” is not supported here (If you don't know what it is, just ignore it). We use another way to define mandatory options and optional options.

Available type options are:

- `s`: strings
- `i`: integers
- `f`: real numbers
- `o`: extended integer, an octal string (0 followed by 0, 1 .. 7), or a hexadecimal string (0x followed by 0 .. 9, A .. F, case insensitive), or a binary string (0b followed by a series of 0 and 1).

Available desttype settings are:

- @: array, allow more than one arguments for an option.
- %: hash, allow arguments like name=value.
- nothing: scalar, single argument for single option.

Available repeat settings are formatted as {\d,\d}. Note there is no blank character inside:

- {2}: exactly 2 arguments for an option.
- {2,}: at least 2 arguments for an option.
- {,4}: at most 4 arguments for an option.
- {2,5}: minimal 2 and maximal 5 arguments for an option.

Note although @ and {\d,\d} are all for array, their usages are different. If option is specified as tag=i@, -tag 1 -tag 2 is only valid. And if option is specified as tag=i{2}, -tag 1 2 is only valid.

Table 1 contains detailed examples for each type of option specification:

Options	Command-line arguments	Value of tag
tag=i	--tag 1	1
	--tag 1 --tag 2	2, only take the last one
	--tag 0.1	Error: Value "0.1" invalid for option tag (number expected)
	--tag a	Error: Value "a" invalid for option tag (number expected)
	--tag	Error: Option tag requires an argument
	no argument	tag is mandatory, please specify it
tag=s	--tag 1	"1". Here double quote is used because it is specified as a string.
	--tag 0.1	"0.1"
	--tag a	"a"
tag=f	--tag 1	1
	--tag 0.1	0.1
	--tag a	Error: Value "a" invalid for option tag (real number expected)
tag=o	--tag 1	1
	--tag 0b001001	9
	--tag 0721	465
	--tag 0xaf2	2802
	--tag 0.1	Error: Value "0.1" invalid for option tag (extended number expected)
	--tag a	Error: Value "a" invalid for option tag (extended number expected)
tag	--tag 1	TRUE
	--tag 0	TRUE, it doesn't care the value for the option.
	--tag 0.1	TRUE
	--tag a	TRUE
	--tag	TRUE
	no argument	FALSE
tag!	--tag	TRUE
	--no-tag	FALSE
tag=i@	--tag 1	1
	--tag 1 --tag 2	c(1, 2)
tag=i%	--tag 1	Error: Option tag, key "1", requires a value
	--tag name=1	tag\$name = 1, tag is a list.
tag=i{2}	--tag 1	Error: Insufficient arguments for option tag
	--tag 1 2	1 2
	--tag 1 --tag 2	Error: Value "-tag" invalid for option tag

Table 1: Detailed example of option specification

4 Set default value and import options as variables

Options will be imported into user's environment as R variables by default. The first option name in option alternative names will be taken as variable name, (e.g. for specification of `length|size=s`, `length` will be used as the variable name.) which means, it must be a valid R variable name. Any definition of these variables in front of `GetoptLong()` will be treated as default values for corresponding options. If options already have default values, they are optional in command-line. If the variable is defined as a function before `GetoptLong()` is called, it is treated as undefined. Please note your option names should not start with the dot. Although it is valid for R variables but it is not allowed for `Getopt::Long` module.

5 Help and version options

`help` and `version` are two universal options. By default, these two options will be inferred from user's source code.

By default, `GetoptLong()` only provides descriptions of all specified options. Users can set the option by `GetoptLong.options('startingMsg')` and `GetoptLong.options('endingMsg')` to add information for a complete help message. And `version` is from `VERSION` variable defined in user's environment (Of course, `VERSION` should be defined before `GetoptLong()`).

```
GetoptLong.options('startingMsg' =  
'An example to show how to use the packages  
' )  
  
GetoptLong.options('endingMsg' =  
'Please contact author@gmail.com for comments  
' )  
VERSION = "0.0.1"  
GetoptLong(...)
```

Then you can specify help:

```
~\> Rscript command.R --help  
An example to show how to use the packages  
Usage: Rscript test.R [options]  
  
--tag integer  
  this is a description of tag which is long long and very long and extremely  
  long...  
  
--help  
  Print help message and exit  
  
--version  
  Print version information and exit  
  
Please contact author@gmail.com for comments
```

Or print version of your script:

```
~\> Rscript command.R --version  
0.0.1
```

6 Configuring `Getopt::Long`

Configuration of `Getopt::Long` can be set by `GetoptLong.options("config")`:

```
GetoptLong.options("config" = "bundling")
GetoptLong.options("config" = c("no_ignore_case", "bundling"))
```

With different configuration, it can support more types of option specifications:

```
-a -b -c -abc
-s 24 -s24 -s=24
```

Please refer to website of `Getopt::Long`(http://search.cpan.org/perldoc?Getopt%3A%3ALong#Configuring_Getopt%3A%3ALong) for more information.

7 Specify path of *Perl* in command line

In some conditions that path of binary *Perl* is not in your `PATH` environment variable and you do not have permission to modify `PATH`. You can specify your *Perl* path from command line like:

```
~\> Rscript test.R -a -b -c -- /your/perl/bin/perl
```

Since arguments following after `--` will be ignored by `Getopt::Long`, we take the first argument next to `--` as the path of user-specified *Perl* path.