Parsing command-line arguments by `Getopt::Long`

Zuguang Gu <z.gu@dkfz.de>

November 27, 2013

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`.

Note: A large part of this vignette is copied or modified from `Getopt::Long` doc page on CPAN. And I cannot guarantee all my understanding on `Getopt::Long` is right. So the original website of `Getopt::Long` is always your best reference.

1 Workflow of the wrapping

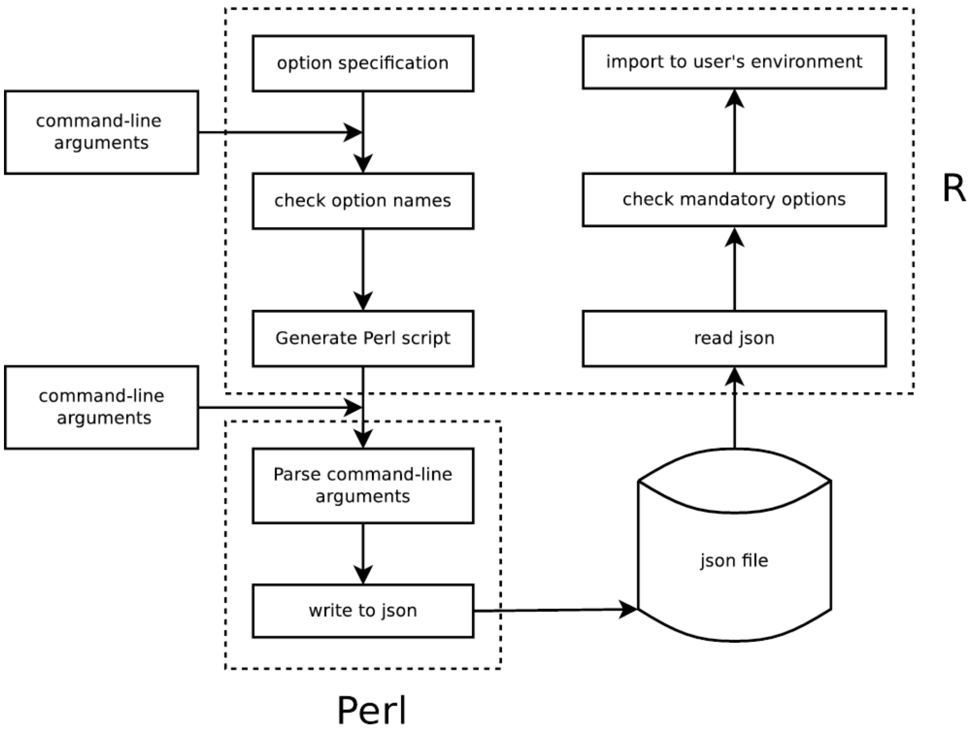

Figure 1: Workflow of wrapping

2 First example

Using `GetoptLong` is simple especially for *Perl* users because the specification is quite similar as in *Perl*.

```
> library(GetoptLong)
> cutoff = 0.05
> GetoptLong(matrix(c(
+     "number=i", "Number of items, integer, mandatory option",
+     "cutoff=f", "cutoff to filter results, optional, default (0.05)",
+     "verbose",  "print messages"
+ ), ncol = 2, byrow = TRUE))
```

Or more simply:

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

The first argument in `GetoptLong` is either a two-column matrix or a simple vector. If it is a simple vector, it should have even number of elements and will be transformed into the two-column matrix by rows internally. In the matrix, the first column is the specification of option names and the second column is the description of corresponding options.

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=s@
```

Here `length|size|l` is a list of alternative names seperated 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 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 uniqe, it will throw an error.

Options for argument specification are:

- no argument specification: taking no argument. Options are logical.
- `!`: taking no argument. Options are logical. You can set its oposite 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.

Please note "`:[desttype]`" is not supported here. We use another way to define mandatory options and optional options. See following sections.

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 one 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.

Following tables are 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
	`--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 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`
	`--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=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, 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. So of course, these already-defined variables should be simple vectors. 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.

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 `options('GetoptLong.startingMsg')` and `options('GetoptLong.endingMsg')` to add informaiton for a complete help message. And version is from `VERSION` variable defined in user's environment (Of course, `VERSION` should be defined before `GetoptLong()`).

```
> options('GetoptLong.startingMsg' = '
+ Usage: Rscript test.R [options]
+ An example to show how to use the packages
+ ')
> options('GetoptLong.endingMsg' = '
+ Please contact author@gmail.com for comments
+ ')
> VERSION = "0.0.1"
> GetoptLong(...)
```

Then you can specify `help`:

```
$~\> Rscript command.R --help

Usage: Rscript test.R [options]
An example to show how to use the packages

  --tag integer
    this is a description of tag which is long long and very long and extremly
    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 `options("GetoptLong.Config")`:

```
> options("GetoptLong.Config" = "bundling")
> options("GetoptLong.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` 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.