Argument Parsing

akwsimmo

February 14, 2022

1 Argument Parsing

Arguments are added to scripts to make them more generalized. Suppose you have a script in which you read from a specific file, but now you want to read from a different file. You can add the filename as an argument to your script, and then you wouldn't have to go find and edit the filename in your script, you would just provide a different filename. Arguments can help to generalize any kind of variables within scripts, not just files. Let's start of with a simple example. Start by creating an argument parser (see `?essentials::ArgumentParser` for more on function `essentials::ArgumentParser`):

```
> parser <- essentials::ArgumentParser(
+     description = "Produce a plot")

[1] "C:/Users/andre/AppData/Local/Temp/RtmpKsMKqC/Rinst2bb874166711"
[1] "essentials"
```

Then, we use `add.argument` to add an argument to the argument parser:

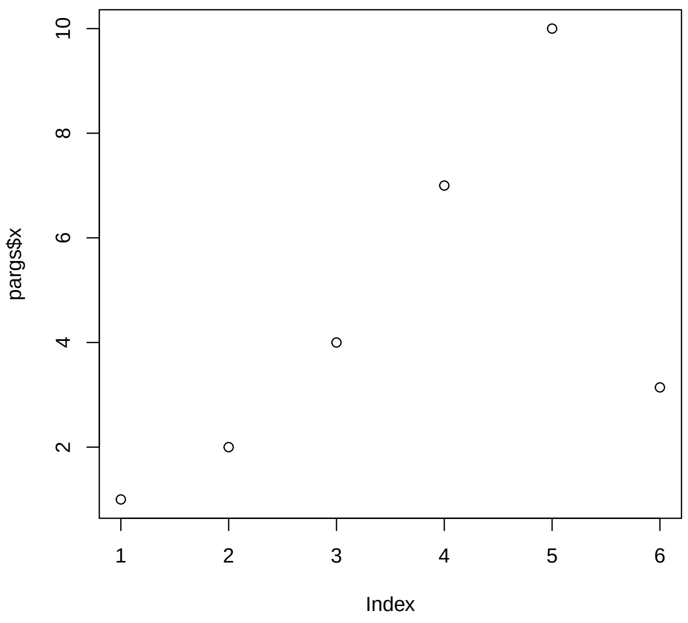
```
> parser$add.argument("x", nargs = "+", type = "numeric",
+     help = "Numbers to plot, provided to argument 'x'",
+     metavariable = "N")
```

Then, parse the arguments the user provided with `parse.args` (normally, argument `args` is not provided, in which case the arguments are taken from the command line arguments, or possibly from `withArgs`):

```
> pargs <- parser$parse.args(
+     args = c("1", "2", "4", "7", "10", "3.1415926535897931"))
```

We should now have a list with element `"x"` which is the numbers we wish to plot. Let's try it:

```
> graphics::plot(x = pargs$x)
```



2 Argument Types

There two types of arguments are positional arguments and named arguments, also called names and flags. A positional argument can be created by:

```
> parser <- essentials::ArgumentParser()
> parser$add.argument("pos-name1", "pos.name2")
> # one or more variable names
```

while a flag can be created by:

```
> parser$add.argument("-a", "--flag_name")
> # one of more variable names starting with one or two hyphens
```

Then, to provide values for these arguments, try:

```
> # any of these are acceptable
> parser$parse.args(c("value1", "-a", "value2"))

Object of class "ParsedArgs"
$flag_name
[1] "value2"

$`pos-name1`
[1] "value1"

> parser$parse.args(c("value1", "-a=value2"))
```

```
Object of class "ParsedArgs"
$flag_name
[1] "value2"

$`pos-name1`
[1] "value1"

> parser$parse.args(c("value1", "--flag_name=value2"))

Object of class "ParsedArgs"
$flag_name
[1] "value2"

$`pos-name1`
[1] "value1"

> parser$parse.args(c("value1", "--flag_name", "value2"))

Object of class "ParsedArgs"
$flag_name
[1] "value2"

$`pos-name1`
[1] "value1"
```

3 Actions

An action specifies what to do when an argument is encountered. `"store"` will store exactly one value, `"append"` stores as many values as are provided, `"store_const"` will return either `default` or `constant` depending upon whether it was provided, `"store_false"`/`"store_true"` are common use cases of `"store_const"` in which `default` is `TRUE/FALSE` and `constant` is `FALSE/TRUE`, and `"count"` will store how many times an argument has been provided.

```
> parser <- essentials::ArgumentParser()
> parser$add.argument("-a", action = "store")
> parser$add.argument("-b", action = "append")
> parser$add.argument("-c", action = "store_const",
+     default = "not provided", constant = "provided")
> parser$add.argument("-d", action = "store_false")
> parser$add.argument("-e", action = "store_true")
> parser$add.argument("-f", action = "count")
> parser$parse.args(
+     c("-a", "1value", "-b", "1st value", "-b", "2nd value", "-ff"))

Object of class "ParsedArgs"
$f
[1] 2

$e
[1] FALSE
```

```
$d
[1] TRUE

$c
[1] "not provided"

$b
[1] "1st value" "2nd value"

$a
[1] "1value"

> parser$parse.args(
+     c("-c", "-d", "-e", "-fff"))

Object of class "ParsedArgs"
$f
[1] 3

$e
[1] TRUE

$d
[1] FALSE

$c
[1] "provided"

$b
character(0)

$a
```

See `?essentials::add.argument` for more information on actions.

4 Choices

When providing arguments that are integers or strings, it is convenient to have a list of possible choices to be provided (choices are available for all types of data, just that other types are far less common use cases). For example:

```
> parser <- essentials::ArgumentParser()
> parser$add.argument("--arg1", choices = c("choice1", "option2", "string3"))
> parser$add.argument("--arg2", choices = 1:10, type = "integer")
> parser$parse.args(c("--arg1=option2", "--arg2=7"))

Object of class "ParsedArgs"
$arg2
[1] 7
```

```
$arg1
[1] "option2"

> parser$parse.args("--arg1=s")  # strings may be abbreviated

Object of class "ParsedArgs"
$arg2


$arg1
[1] "string3"

> tryCatch({
+     parser$parse.args(c("--arg2", "-8"))
+ }, condition = print)

<simpleError in check.args(): '--arg2' should be one of "1", "2", "3", "4", "5", "6", "7",
```