formatR

Format R code automatically

Yihui Xie / 2017-12-07

1. Installation

You can install **formatR** from CRAN, or XRAN if you want to test the latest development version:

```
install.packages("formatR", repos = "http://cran.rstudio.com")
#' to install the development version, run
#' install.packages('formatR', repos = 'https://xran.yihui.name')
```

Or check out the Github repository and install from source if you know what this means. This page is always based on the development version.

```
library(formatR)
sessionInfo()
```

```
## R version 3.4.2 (2017-09-28)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.1
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dy
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  base
##
## other attached packages:
## [1] formatR_1.5
##
## loaded via a namespace (and not attached):
## [1] compiler_3.4.2  magrittr_1.5    tools_3.4.2     stringi_1.1.6
## [5] knitr_1.18      methods_3.4.2   stringr_1.2.0   evaluate_0.10.1
```

2. Reformat R code

The **formatR** package was designed to reformat R code to improve readability; the main workhorse is the function `tidy_source()`. Features include:

- long lines of code and comments are reorganized into appropriately shorter ones
- spaces and indent are added where necessary
- comments are preserved in most cases
- the number of spaces to indent the code (i.e. tab width) can be specified (default is 4)
- an `else` statement in a separate line without the leading `}` will be moved one line back
- `=` as an assignment operator can be replaced with `<-`
- the left brace `{` can be moved to a new line

Below is an example of what `tidy_source()` can do. The source code is:

```
## comments are retained;
# a comment block will be reflowed if it contains long comments;
#' roxygen comments will not be wrapped in any case
1+1

if(TRUE){
x=1  # inline comments
}else{
x=2;print('Oh no... ask the right bracket to go away!')}
1*3 # one space before this comment will become two!
2+2+2    # only 'single quotes' are allowed in comments

lm(y~x1+x2, data=data.frame(y=rnorm(100),x1=rnorm(100),x2=rnorm(100)))  ### a line
1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1  # comment after a long line
## here is a long long long long long long long long long long long long long comm
```

We can copy the above code to clipboard, and type `tidy_source(width.cutoff = 50)` to get:

```
## comments are retained; a comment block will be
## reflowed if it contains long comments;
#' roxygen comments will not be wrapped in any case
1 + 1

if (TRUE) {
    x = 1  # inline comments
} else {
    x = 2
    print("Oh no... ask the right bracket to go away!")
}
1 * 3  # one space before this comment will become two!
2 + 2 + 2  # only 'single quotes' are allowed in comments

lm(y ~ x1 + x2, data = data.frame(y = rnorm(100), x1 = rnorm(100),
    x2 = rnorm(100)))  ### a linear model
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
    1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1  # comment after a long line
## here is a long long long long long long long long
## long long long long long comment which will be
## wrapped
```

Two applications of `tidy_source()`:

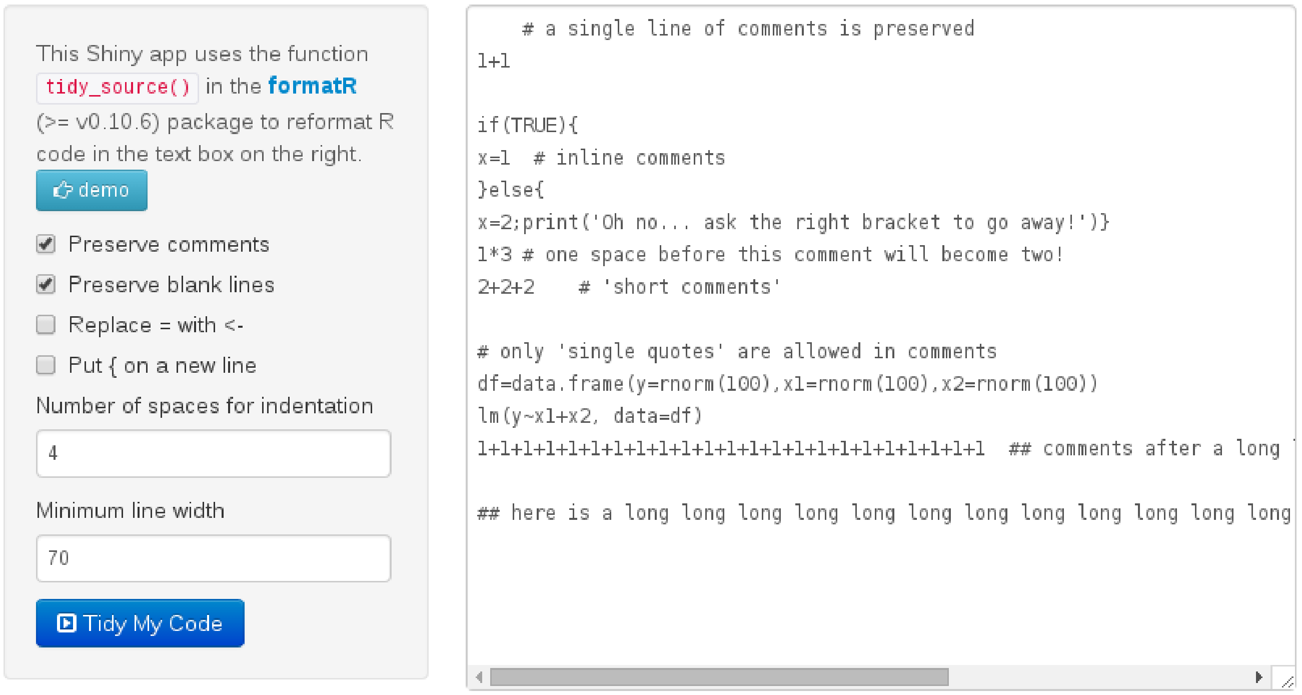
- `tidy_dir()` can reformat all R scripts under a directory
- `usage()` can reformat the usage of a function, e.g. compare `usage()` with the default output of `args()`:

```
library(formatR)
usage(glm, width = 40)  # can set arbitrary width here
## glm(formula, family = gaussian, data,
##     weights, subset, na.action,
##     start = NULL, etastart, mustart,
##     offset, control = list(...),
##     model = TRUE, method = "glm.fit",
##     x = FALSE, y = TRUE,
##     contrasts = NULL, ...)
args(glm)
## function (formula, family = gaussian, data, weights, subset,
##     na.action, start = NULL, etastart, mustart, offset, control = list(...),
##     model = TRUE, method = "glm.fit", x = FALSE, y = TRUE, contrasts = NULL,
##     ...)
## NULL
```

3. The Graphical User Interface

If the **shiny** packages has been installed, the function `tidy_app()` can launch a Shiny app to reformat R code like this (live demo):

```
formatR::tidy_app()
```



This Shiny app uses the function `tidy_source()` in the **formatR** (>= v0.10.6) package to reformat R code in the text box on the right.

- [✓] Preserve comments
- [✓] Preserve blank lines
- [ ] Replace = with <-
- [ ] Put { on a new line

Number of spaces for indentation: 4

Minimum line width: 70

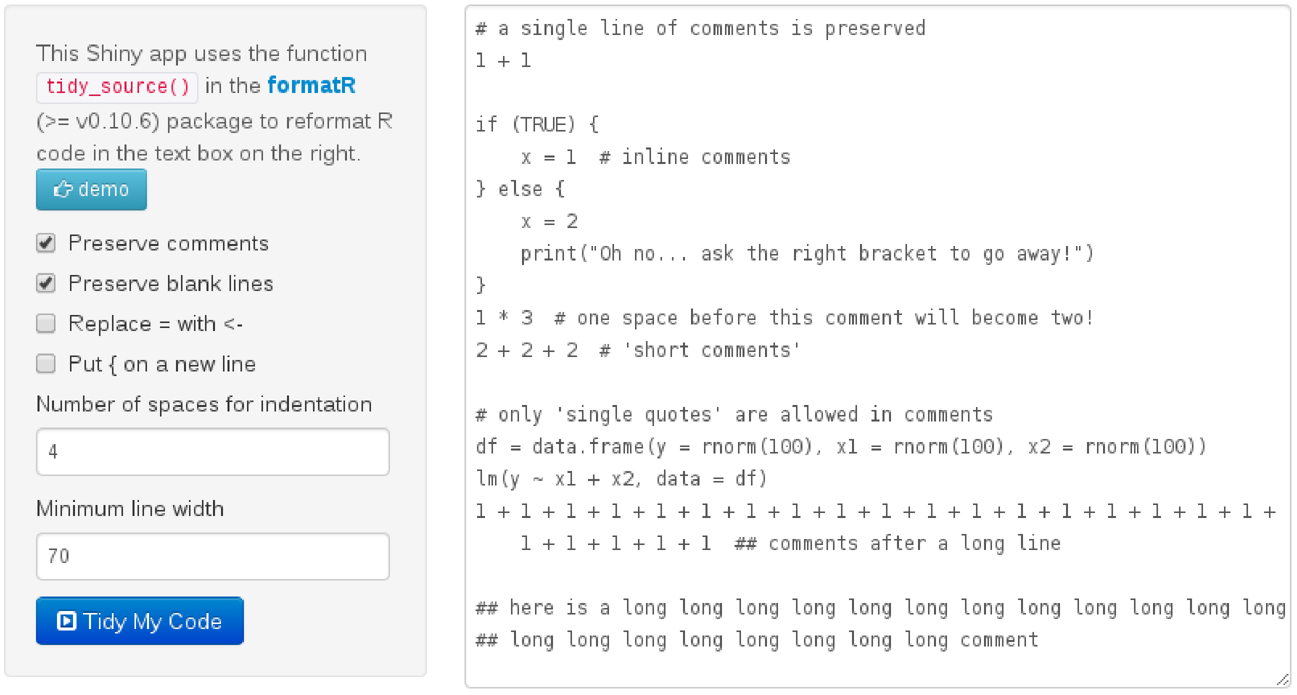
```
    # a single line of comments is preserved
1+1

if(TRUE){
x=1  # inline comments
}else{
x=2;print('Oh no... ask the right bracket to go away!')}
1*3 # one space before this comment will become two!
2+2+2    # 'short comments'

# only 'single quotes' are allowed in comments
df=data.frame(y=rnorm(100),x1=rnorm(100),x2=rnorm(100))
lm(y~x1+x2, data=df)
1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1  ## comments after a long

## here is a long long long long long long long long long long long long
```

After hitting the `Tidy` button:


This Shiny app uses the function `tidy_source()` in the **formatR** (>= v0.10.6) package to reformat R code in the text box on the right.

- [✓] Preserve comments
- [✓] Preserve blank lines
- [ ] Replace = with <-
- [ ] Put { on a new line

Number of spaces for indentation: 4

Minimum line width: 70

```
# a single line of comments is preserved
1 + 1

if (TRUE) {
    x = 1  # inline comments
} else {
    x = 2
    print("Oh no... ask the right bracket to go away!")
}
1 * 3  # one space before this comment will become two!
2 + 2 + 2  # 'short comments'

# only 'single quotes' are allowed in comments
df = data.frame(y = rnorm(100), x1 = rnorm(100), x2 = rnorm(100))
lm(y ~ x1 + x2, data = df)
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
    1 + 1 + 1 + 1 + 1  ## comments after a long line

## here is a long long long long long long long long long long long long
## long long long long long long long long comment
```

4. Evaluate the code and mask output in comments

It is often a pain when trying to copy R code from other people's code which has been run in R and the prompt characters (usually >) are attached in the beginning of code, because we have to remove all the prompts > and + manually before we are able to run the code. However, it will be convenient for the reader to understand the code if the output of the code can be attached. This motivates the function `tidy_eval()`, which uses `tidy_source()` to reformat the source code, evaluates the code in chunks, and attaches the output of each chunk as comments which will not actually break the original source code. Here is an example:

```r
set.seed(123)
tidy_eval(text = c("a<-1+1;a  # print the value", "matrix(rnorm(10),5)"))
```

```
a <- 1 + 1
a  # print the value
## [1] 2

matrix(rnorm(10), 5)
```

```
##          [,1]    [,2]
## [1,] -0.56048  1.7151
## [2,] -0.23018  0.4609
## [3,]  1.55871 -1.2651
## [4,]  0.07051 -0.6869
## [5,]  0.12929 -0.4457
```

The default source of the code is from clipboard like `tidy_source()`, so we can copy our code to clipboard, and simply run this in R:

```
library(formatR)
tidy_eval()
# without specifying any arguments, it reads code from clipboard
```

5. Showcase

We continue the example code in Section 2, using different arguments in `tidy_source()` such as `arrow`, `blank`, `indent`, `brace.newline` and `comment`, etc.

Replace = with <-

```
if (TRUE) {
    x <- 1  # inline comments
} else {
    x <- 2
    print("Oh no... ask the right bracket to go away!")
}
```

Discard blank lines

Note the 5th line (an empty line) was discarded:

```
## comments are retained; a comment block will be reflowed if it
## contains long comments;
#' roxygen comments will not be wrapped in any case
1 + 1
if (TRUE) {
    x = 1  # inline comments
} else {
    x = 2
    print("Oh no... ask the right bracket to go away!")
}
1 * 3  # one space before this comment will become two!
```

Reindent code (2 spaces instead of 4)

```
if (TRUE) {
  x = 1  # inline comments
} else {
  x = 2
  print("Oh no... ask the right bracket to go away!")
}
```

Move left braces { to new lines

```
if (TRUE)
{
    x = 1  # inline comments
} else
{
    x = 2
    print("Oh no... ask the right bracket to go away!")
}
```

Discard comments

```
1 + 1
if (TRUE) {
    x = 1
} else {
    x = 2
    print("Oh no... ask the right bracket to go away!")
}
1 * 3
2 + 2 + 2
lm(y ~ x1 + x2, data = data.frame(y = rnorm(100), x1 = rnorm(100),
    x2 = rnorm(100)))
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
    1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1
```

6. Further notes

The tricks used in this packages are very dirty. There might be dangers in using the functions in **formatR**. Please read the next section carefully to know exactly how comments are preserved. The best strategy to avoid failure is to put comments in complete lines or after *complete* R expressions. Below are some known cases in which `tidy_source()` fails.

In-line comments after an incomplete expression or ;

```
1 + 2 + ## comments after an incomplete line
    3 + 4
x <- ## this is not a complete expression
     5
x <- 1; # you should not use ; here!
```

It is not a good idea to interrupt R code with comments and sometimes it can be confusing – comments should come after a complete R expression naturally; by the way, `tidy_source()` will move the comments after { to the next line, e.g.

```
if (TRUE) {## comments
}
```

will become

```
if (TRUE) {
    ## comments
}
```

Inappropriate blank lines

Blank lines are often used to separate complete chunks of R code, and arbitrary blank lines may cause failures in `tidy_source()` as well when the argument `blank = TRUE`, e.g.

```
if (TRUE)

{'this is a BAD style of R programming!'} else 'failure!'
```

There should not be a blank line after the `if` statement. Of course `blank = FALSE` will not fail in this case.

`?` with comments

We can use the question mark (?) to view the help page, but **formatR** package is unable to correctly format the code using `?` with comments, e.g.

```
?sd  # help on sd()
```

In this case, it is recommended to use the function `help()` instead of the short-hand version `?`.

`!!` and `!!!` from the rlang package

The syntactic shortcuts `!!` and `!!!` from the **rlang** packages will be ruined by `tidy_source()`, e.g.,

```
rlang::quo(mean(!! 1:10 * 2))
```

will be reformatted as:

```
rlang::quo(mean(!(!1:10 * 2)))
```

`->` with comments

We can also use the right arrow `->` for assignment, e.g. `1:10 -> x`. I believe this flexibility is worthless, and it is amazing that a language has three assignment operators: `<-`, `=` and `->` (whereas almost all other languages uses `=` for assignment). Bad news for **formatR** is that it is unable to format code using both `->` and comments in a line, e.g.

```
1:10 -> x  # assignment with right arrow
```

I recommend you to use `<-` or `=` consistently. What is more important is consistency. I always use `=` because it causes me no confusion (I do not believe it is ever possible for people to interpret `fun(a = 1)` as assigning `1` to a variable `a` instead of passing an argument value) and `<-` is more dangerous because it works everywhere (you might have unconsciously created a new variable `a` in `fun(a <- 1)`; see an example here). The only disadvantage is that most R people use `<-` so it may be difficult to collaborate with other people.

The pipe operator `%>%`

Although `tidy_source()` won't ruin your code that contains the pipes, you won't be happy with it: your line breaks after the pipes won't be preserved. See #54.

7. How does `tidy_source()` actually work?

In a nutshell, `tidy_source(text = code)` is basically `deparse(parse(text = code))`, but actually it is more complicated only because of one thing: `deparse()` drops comments, e.g.,

```
deparse(parse(text = "1+2-3*4/5 # a comment"))
```

```
## [1] "expression(1 + 2 - 3 * 4/5)"
```

The method to preserve comments is to protect them as strings in R expressions. For example, there is a single line of comments in the source code:

```
  # asdf
```

It will be first masked as

```
invisible(".IDENTIFIER1  # asdf.IDENTIFIER2")
```

which is a legal R expression, so `base::parse()` can deal with it and will no longer remove the disguised comments. In the end the identifiers will be removed to restore the original comments, i.e. the strings `invisible(".IDENTIFIER1` and `.IDENTIFIER2")` are replaced with empty strings.

Inline comments are handled differently: two spaces will be added before the hash symbol `#`, e.g.

```
1+1#  comments
```

will become

```
1+1  #  comments
```

Inline comments are first disguised as a weird operation with its preceding R code, which is essentially meaningless but syntactically correct! For example,

```
1+1 %InLiNe_IdEnTiFiEr% "#  comments"
```

then `base::parse()` will deal with this expression; again, the disguised comments will not be removed. In the end, inline comments will be freed as well (remove the operator `%InLiNe_IdEnTiFiEr%` and surrounding double quotes).

All these special treatments to comments are due to the fact that `base::parse()` and `base::deparse()` can tidy the R code at the price of dropping all the comments.

8. Global options

There are global options which can override some arguments in `tidy_source()`:

argument	global option	default
`comment`	`options('formatR.comment')`	`TRUE`
`blank`	`options('formatR.blank')`	`TRUE`
`arrow`	`options('formatR.arrow')`	`FALSE`
`indent`	`options('formatR.indent')`	`4`
`brace.newline`	`options('formatR.brace.newline')`	`FALSE`

Also note that single lines of long comments will be wrapped into shorter ones automatically, but roxygen comments will not be wrapped (i.e., comments that begin with `#'`).