Bulding\_tidy\_tools

Jian (AKA Frank)

06/05/2019

Table of Contents

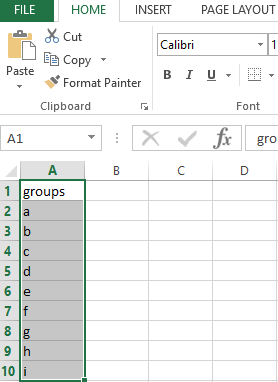
## Introduction

I am very futunate to attended a two days workshop which taught by Hadley Wickham early this month. The workshop is around building tidy tools to automated repeated works. It was a hands on pratical workshop. However, in my opionion, the concept behind the tidy tools is building a tidy mind. A tidy mind to modulised all the critical part of a data analysis workflow, and clarify the necessary tools to achieve the goal in each module. Therefore, I believe the workshop can benefit our work no matter which level of R users you believe youselves are. In a way, we are all beginners to learn new tips and tricks. I summrised 3 parts that I think the most useful ones.In case of personal bias, please find the full materials [here](rstd.io/tidytools19).

## Shortcut keys are useful (for Rstudio users)

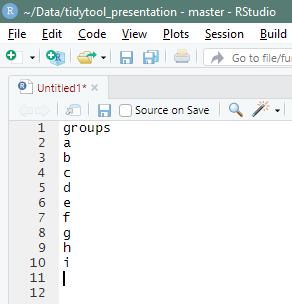
This might not be true when people first try to memory so many shortcut keys. especially when point and click is so easy by the mouse. However, it is a long term benefit. There are some cases, a shortcut key can be very helpful. Here is one example:

Imaging you have a few group factors that are stored in column-wise in a excel sheet or in other format.



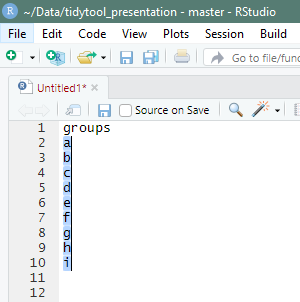
*Demo\_Step\_1*

Turn them into a vector manually can be painful.



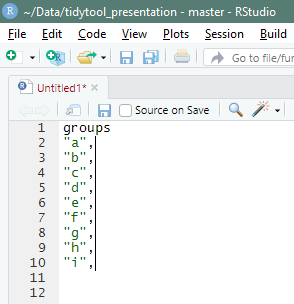
*Demo\_Step\_2*

**[alt + drag]** in Rstudio can be useful. Press **alt** and then **drag down** with the mouse to trigger the multi-cursors.

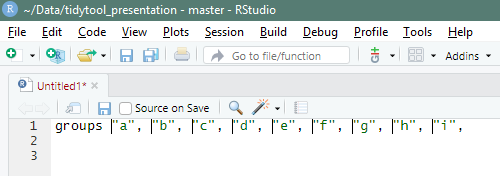


*Demo\_Step\_3*

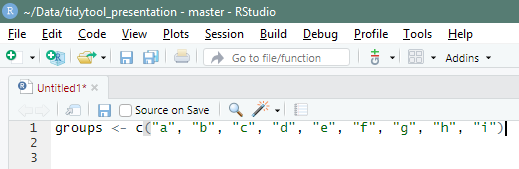
And then, edit the string as normal.



*Demo\_Step\_4*



*Demo\_Step\_5*



*Demo\_Step\_6*

Other useful shortcut keys you probably already know are:

**[Ctrl + enter]** Run the selected line.

**[Ctrl + .]** **[Ctrl + O]** navigating files inside a R project. The first one is recommend because users only need to remember the file name if the file inside your R project.

**[Ctrl + shift + N]** open a new file

**[alt + Ctrl + K]** shortcut key reference page

## Achieve something small first

In the past, the definition of a R function for me was one function can at least do one or two jobs at once, such as reading data following a cleaning process Sometime, I even wrote a few hundred lines code to do the whole data analysis workflow. This is only because I was lucky enough to handle the consistent modelling data. However, it was extremely hard to come up with a good testing procedure to test the whole script. For a while, I convinced myself that I am the most dummy person in the world who cannot even figure out how to do unit testing on my own scripts. Now I realized that my mistakes were: 1. Misunderstood the idea of function programming; 2. Misunderstood the entire idea of unit testing.

Try to achieve everything in one go is the ultimate idea but never is the first step. Function programming in R is a good example. The first step to write a function in R should be always making one single call working first. For instance,

unique(mtcars$cyl)

## [1] 6 4 8

and then to identify the pattern by copying and pasting.

unique(mtcars$cyl)  
unique(mtcars$am)  
unique(mtcars$hp)

Finally, sick of copy and paste. the first for loop is born.

for (i in seq\_len(length(mtcars))){  
 x <- unique(mtcars[[i]])  
 print(x)  
}

The for loop was working perfect fine for simple tasks till one day there is a large data set and lots of computation requires. Dr.Google suggests better use \*apply or map rather than for loop in R. The code got upgraded.

lapply(mtcars, function(x) unique(x))  
purrr::map(mtcars, ~ unique(.x))

I just couldn’t help myself to add more calls once I got familiar with this strategy. Thinking “wow, I can add a function call here to achieve more - that makes sense”. Things get complicated quickly when more functionalities added. It often turns out that I didn’t achieve much by writing a giantic piece of code, only long waiting time to see output which always has suprises in it. For example, I wrote a script that reads, transforms, summarises and visulisie the modelling data all at one go. It broken easily when I just adjusted one parameter to change a colour of a point. The waiting time got longer and the logic got messer.

Complexcity shouldn’t be the goal of functional programming in R, isntead, simplicity is. One thing I learned in this workship is simplify as much as you can for one function call. One R function should only achieve one thing at a time, and then use a bunch of functions to solve a complex issue. As a lego fun, I now would like to think each function (either customised or pre-defined) is a brick. I should use various bricks to **build** a object rather than using a giant brick to curve out a object.

Hadley said in the workshop “It is probably the sign you should break the code into smaller pieces when you can’t write a simple unit testing to test the code.”

Unit testing haunted me for at least a year till now. I finally realised that I am not the dummiest one. The only reason I couldn’t test my script was because it is too long and too complicated. The script I tried to test is way beyond the coverage of unit testing since it is almost like a whole workflow testing. It is not impossible to do testing on the whole data analysis workflow. The chanllenges are the definition of the mimumun dataset for testing, dependencies of analysis method and variances in different projects. Some experters are looking into this space now, <https://github.com/ropensci/rrrpkg>. No ready-to-use tool yet.

For any R users, we probably do testing all the time. For example, checking function output in console, browsing data by calling str or View. Hoenstley, all the codes we wrote probably can be called “**T**est **D**riven **D**evelopment”(TDD) since we need to make sure that the output on previous step is the one for next computation. The only true different is human test is error prone and might not as reliable as machine automated testing, at least for me. A small example here to illustrate human “TDD”. Image I’d like to have a function that adds 1 everytime I pass an interger to it. The quickest and simpliest solution probably something like this:

add\_one <- function(x) x + 1  
add\_one(1)

## [1] 2

It seems that the function just worked. I added another line to see the type of the output because I deseparately need the output to be interger.

typeof(add\_one(1))

## [1] "double"

NO. It is not an interger. So I probably need to go back and wrap the x and 1 by calling as.interger. Alternatively, I could do a real **Test driven development**.

library(testthat)  
expect\_type(add\_one(1), "integer")

## Error: add\_one(1) has type `double`, not `integer`.

Appreantly, it won’t work since I haven’t define the function. Define the dummy version again for illustration.

add\_one <- function(x) x + 1  
expect\_type(add\_one(1), "integer")

## Error: add\_one(1) has type `double`, not `integer`.

The testing tells me that the type is double. So I could specific the type.

add\_one <- function(x) as.integer(x) + as.integer(1) # will be double if not define as integer.  
  
expect\_type(add\_one(2), "integer") # Return nothing means pass  
expect\_equal(add\_one(1), 2)   
expect\_error(add\_one("a"), "x must be an integer") # Should be an error if x is a non interger value

## Warning in add\_one("a"): NAs introduced by coercion

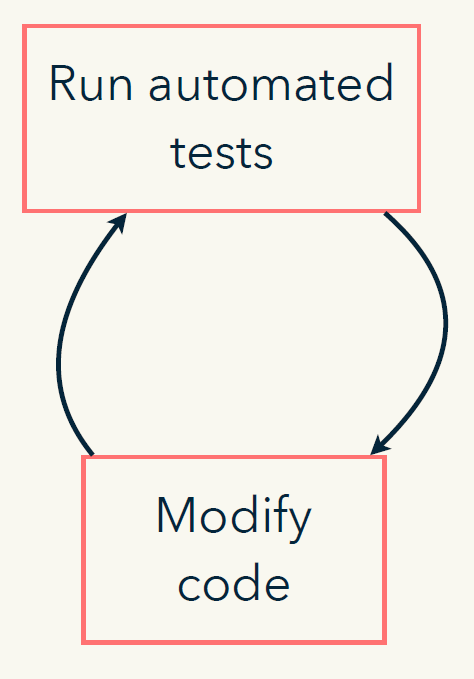
## Error: `add\_one("a")` did not throw an error.

The simple function does not report error. So I could modify it into a more robust stage.

add\_one <- function(x) {  
 stop(is.integer(x),  
 call. = FALSE,  
 "x must be an integer")  
 as.integer(x) + as.integer(1)   
}  
expect\_error(add\_one("a"), "x must be an integer")

Testing is everywhere whenever we use R to process data. Sometimes it is sufficient that testing the output by eyes in the console. For example, just want to check if the data structure layout. Sometimes a tested code can save lots of debugging time for the futrue. As the dummy TDD example shows, each step of the testing has been recorded, therefore, we know exactly what has been tested and do not need to waste time on checking the tested ones. Unit testing is not about predict all the possibilies in the future. I believe it is quite the opposite, recording the past.

Figure.1 shows the concept of unit testing.

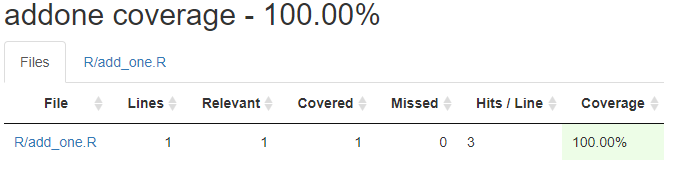


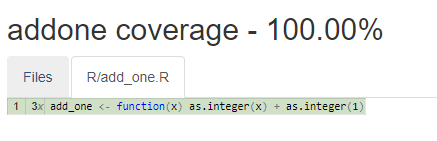
*Figure.1 Snapshot from the Material - 03-test*

The first couple of functions done by this strategy will probably painful since the back and forward from the actual function file to the testing file. Yes, the testing script is better to be separated with the actual function file and they are better to be stored in a package project. For more details about setting things up and do this process automatically, please refers to the materials or give me a call.

Once we pass the fiddling starting stage, everything will be awesome. Achieve something small and then scale it up.

To visulise our achievement, use devtools::test\_coverage() to find out how many tests have been down and the coverage of those tests.





One important thing is the percentage coverage is not a goal to chase. It does not have to be 100% all the times. Time is too precious to be wasted on testing less important part. There is no golden answer to evulate which script is worth which one is not. It depends on the resusable value of the script which can only be figured out by time. I will definitely write unit testing for a function that I reused in multiply scripts. The code for a critical analysis should also be tested in order to have the exact same results for future run. Other suitations probably depends on personal choices. Have time in the weekend? Write some unit testings just for pratising.

## Be nice to yourself

I hate my younger self more than anyone else. This is because two reasons:

1. He is an tight ass in terms of commenting the codes, either very few broken sentences or none words at all;
2. He made my life couldn’t be harder because the inconsistences in his codes.

To make my futrue life a bit easier, I believe that three improvement can be made:

1. Document the function that has been reused a lot;
2. Use meaningful names when writting functions;
3. Thinking carefully before damping times to develop a function that is already existing in the R community.

### Improvenment one

Documenting R code is probably the most chanllenge part for me in the last 2.5 years. Even the simply strategy, commenting beside each line of code, seems so chanllegning. Sometime, I thought there would be no body else use this script anymore so why bother to put any comment. It turns out that I had to reuse the script a few weeks later and had to figure out why did I do this and that. Sometime, I was under pressure to get stuff done and didn’t have time to put thoughtful words in the script. It turns out that I’m always holding this same excuse and bullying my future self by unexplained code. Sometimes, I just didn’t know how to document the code because I didn’t fully understand it myself. It turns out that the only chance to get to understand some complex functions is to document it well in my own words.

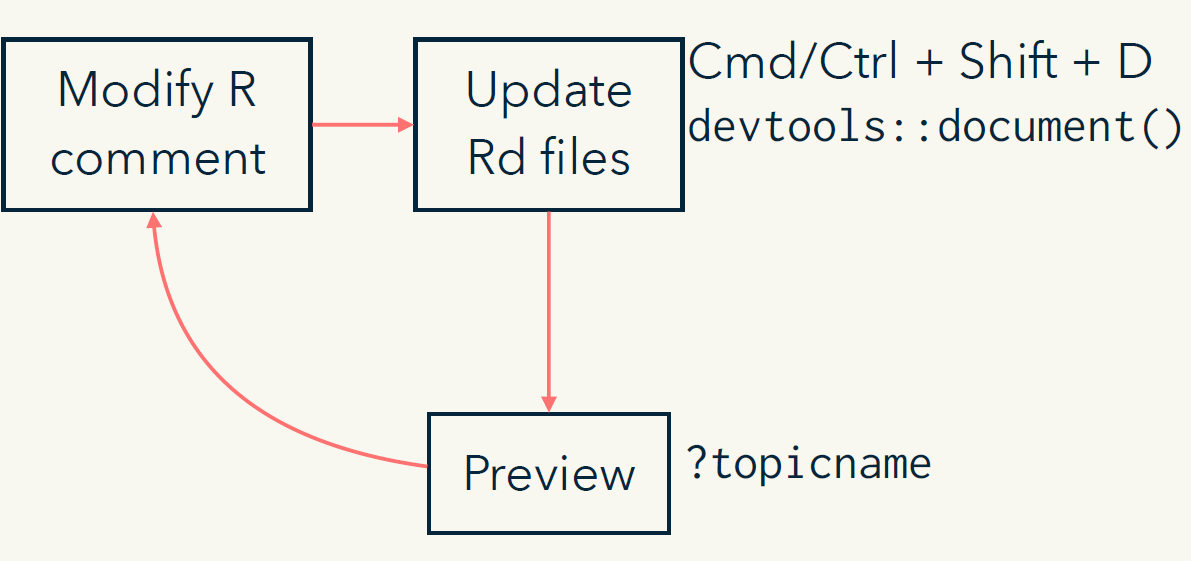
The most common documentation in R is probably commenting beside the code by using #. This is good for a quick solution and short term use functions. However, it might be a bit messy and confusing if the script gets too complex. For a long term benefit, using roxygen2 package is an ideal solution. Documentation workflow

**[Crtl + Alt + Shift + R]** is the shortcut key to insert a Roxygen skeleton. The package has no built in help documentation because it contains no functions. The best place to understand the syntax and implemenataion is in [Hadley’s book](http://r-pkgs.had.co.nz/man.html#roxygen-comments).

A simple example of documenting a function:

#' add\_one  
#'  
#' @param x an interger  
#'  
#' @return interger(s)  
#' @export   
#'  
#' @examples  
#' add\_one(1)  
  
add\_one <- function(x) {  
 stop(is.integer(x),  
 call. = FALSE,  
 "x must be an integer")  
 as.integer(x) + as.integer(1)   
}

Figure 2 demonstrates the workflow of docomenting functions automaticlly. Full details can be found either in the materials or [the book](http://r-pkgs.had.co.nz/man.html#man-functions)



*Figure.2 Snapshot from the Material - 09-document*

### Improvenment two

Rstudio users can use the shortcut key **[Ctrl + .]** (Not working in powerplant version) to find the file without going through all the directories. This means that we only need to remember the file names that have the function or other files we need. Inconsistent and less informative names can be triky to memorise. For example, the base R function regexpr and grepgexpr are very diffcult to understand their functionalities till one look the help documentation in details. These functions are quite useful but perphas too creative for a non-programmer user. Thus, it will save the future self if one put a bit more thought on naming a function. Two suggestions from the workshop:

1. Use **verb** to name the function that do things;
2. Use **noun** to name intermidate product.

### Improvenment three

Why bother to spend heaps of time to develop something that is already existing and well deveoped. It might be sounds cool to develop everything on my own. However, there is a great cost on this, time. I am only an amture R programmer and I use R to help me understand the data. I have the intention to make R better but not now.

## Acknowledgement

Thanks a lot for the support from Linley, Paul, Edmar, Jo and Adrain. I couldn’t get this far on my R jounery without your helps.