Summary of some "best practices" for R
--Dan Rabosky

1. Executing R code. ***Always use an editor (like the built-in R editor) for anything other than the simplest 1-line statements.*** This saves your work, and it doesn't choke when executing multiple lines. On a mac, you can simply execute code from the R editor by highlighting code in the editor window and pressing "command + return". If executing code line-by-line, command+enter will execute the line of code indicated by your cursor in the editor window.

NEVER paste code from editor window into R console!

2. The "equals" sign in R is the arrow operator. <-
This must always go the same direction:

```
x <- 10
```

and never

```
10 -> x
```

The latter will work, but makes virtually unreadable code.

3. In general, *be consistent!* Be consistent in how you format your code, particularly with indentation and spacing. The advantages of this will become more obvious the longer you use R.

4. Indentation. The concept of something "belonging" to something else, e.g when a block of code "belongs" to a function, or to a "loop". ALWAYS indent one level for each such nested set of statements. This is done either with a tab space or with 3 single spaces. **Must be done consistently**.

4. Curly braces versus square brackets versus parentheses. Learn the difference between these – they are very different things!

5. With curly braces, you must use them consistently. My preference is:

```
myfunction <- function(x, y){
   # some block of code here:
   # more code
}
```

6. You "comment out" lines of code with the # symbol. Any line of code preceded by this symbol will be ignored by R.

7. Learn the difference between (i) default objects in your R workspace, like built-in functions (e.g, `rnorm`, `read.table`, and others); (ii) Objects and stuff that you add by loading libraries (e.g., the `read.tree` function from the *ape* package); and (iii) data and objects that you create. You can use the functions `ls` to view all the objects that you have added to your workspace (e.g., all the stuff that falls into category (iii) above), and function `rm` will remove anything from your workspace.

8. Every time you do a new analysis, set up new script files to save all your work. Just title them well; even create separate directories for things. People end up with huge messes by trying to do all sorts of stuff in single files, when there is no cost to just having separate scripts for everything and everything is much cleaner. ORGANIZE this - don't just have a giant pileup of scripts on the Desktop.

9. The easiest way to work through something, or to debug something, is to go through line-by-line. For whatever reason, many students have an impulse to run all their code at once, all the time. Often, something will fail on the first line, but if you try to run 50 lines of code at once, you'll have no idea why it fails.

Never execute multiple lines of code until you have confirmed that everything up to a certain point works. I even do this within loops, to make sure the loops work before I run them for real:

So, if I have:

```
for (i in 1:100){

     Do some simulation
     Save some results
     Do something with results

}
```

What I have my students do, until they get the hang of it, is to first comment out the start and end of the loop:

```
#for (i in 1:100){

     Do some simulation
     Save some results
     Do something with results

#}
```

And then, we set `i = 1` and just execute the core block of code (as in: you only run the yellow highlighted code):

```
#for (i in 1:100){
     i <- 1
     Do some simulation
     Save some results
     Do something with results

#}
```

So now you'll know whether the 'within-loop' part of the algorithm works. This will save you tons of time if you start doing this. When you’ve confirmed that this works for i = 1, you can uncomment it, delete the i = 1, and run the whole block of code:

```
for (i in 1:100){

     Do some simulation
     Save some results
     Do something with results

}
```

10. Whenver you do anything complicated: CONFIRM that code for part A works before moving on to execute more code in part B, whenever B depends on A ! Do this OFTEN, and RELIGIOUSLY. Many students try running multi-line code without ever checking to see if stuff is working. So: let's suppose you are trying to read a data table and do something with it, like this:

```
myData <- read.table(file = 'mydatafile.txt')
DoSomething(myData)
```

My R beginners will do the first operation (reading the data table) and then move on to the second without ever checking to see if the first operation worked correctly. This could very easily happen in the above example, if you used (say) a comma as opposed to tab delimiter with `read.table()`. Then you’ll send a misformatted data table to `DoSomething()`, and it will choke. But the real error occurs if you assume (without checking) that the problem lies with `DoSomething` and not with `myData`. Since the problem is that the data are not read correctly, no amount of debugging the second statement (`DoSomething`) is going to fix the problem! This could very easily be solved if you get in the habit of checking things. E.g, if you are reading a datatable, maybe you know it should be multiple columns, or that it has a header row, so - after

```
myData <- read.table(file = ....)
```

in the editor window, get into the habit of going immediately to the R console to do some checks, like:

```
head(myData)
tail(myData)
dim(myData)
names(myData)
```

and so on, before you ever go to the next step. You should look up the help pages on those functions to see what they do (e.g., `?head`). Other checks will depend on what the you are doing. `length()` is a good check. If working with a vector, you can go to the R console and look at the length or the first few elements.

11. When you have cryptic problems, clear your R workspace with

```
rm(list = ls())
```

Or restart R and **DO_NOT** save your workspace (a bad habit in general). Also, it is a good idea to always to start a fresh analysis with

```
rm(list = ls())
```

at the head of your script file. Otherwise, you can end up with all sorts of garbage in your workspace (and in your analyses!).

12. **Again, never save your workspace, and clear it before beginning a new analysis!** If you are doing complicated analyses, get in the habit of writing your results to files as you move along; then you can always pick up where you left off by reading in the data/results files. *This is much, much safer than trying to save a workspace!*

13. Many students have an aversion to errors. I think new programmers get the feeling that they are breaking their computer whenever something chokes. They need to realize that things fail all the time, and errors are normal and fundamental to making robust code.

14. **If something "works", it doesn't mean it works!** I come back to this because it is so important and typically lost on beginning R students. If you execute code and if it doesn't give an error, you cannot assume that everything is OK! You need to check frequently to make sure that *whatever you think is happening* is *whatever is actually happening*.

15. **ALWAYS use a space to separate variables, operators, and so on**. Some R beginners tend not to use spaces between operators and variables, such that code might look like this:

```
(x<=1&&y>=5)||(y<=2)
```

This code above actually mixes multiple (important) separate statements, and should look like this:

```
(x <= 1 && y >= 5) || (y<=2)
```

You should also put a single space between the assignment operator and whatever you are assigning etc, like:

```
x <- 5
```

NOT

```
x<-5
```

16. Learn the basics of navigating between directories ("folders"), viewing their contents, and setting directories. These operations are accomplished with the "command + D" on a mac, and with the R functions: `setwd, getwd, file.choose, and dir`.