Week 0 Exercise: Introduction to Git, GitHub, and RStudio

Z620: Quantitative Biodiversity, Indiana University

January 6, 2017

OVERVIEW

In this pre-class exercise, you will be introduced to the computing environment that we will be using in Quantitative Biodiversity. By the end of this exercise you will understand the basics of version control and how it is implemented using Git and GitHub, which will require elementary proficiency in UNIX. Last, you will become acquainted with RStudio. You will learn how to perform basic operations and become familiar with how to work with vectors, a one-dimensional array of data.

1) VERSION CONTROL

What is version control?

Version control is an approach to writing text, managing data, and developing computer code that allows users the ability to examine, comment on, and revert back to changes within the entire life of a document, and without being tied to any single computer. In addition, version control allows multiple people in remote locations to collaborate on the same text, code, and data without losing or overwriting any changes.

Why is version control important?

Do you email versions of renamed files with your collaborators? Do you fear that data on a hard drive could get lost or damaged? Do you have a file somewhere that looks like this: "Dissertation-Project-StatsCode-Final-v23.R"? No need to worry. Version control can help you better manage changes to data, manuscripts, and computer code. Thougth this, you can become a more organized and responsible scientist. If that isn't enough, funding agencies and many journals now require authors to provide curated data and can call on authors to provide proof of reproducibility.

How version control works, in a nutshell

Version control works by centralizing a project in a repository (a.k.a., **repo**) located on a server (e.g., a computer connected to the internet). The individual user never directly edits the code, data, or text in this online repo. Instead, the user makes changes to a local version of the project (e.g., on your laptop) and **pushes** those changes to the online version. The entire history of the online version is tracked and protected. Likewise, if the online version is modified by a collaborator, then the user merely **pulls** in the changes from the online version. All this pushing, pulling, and deciding how different versions from different computers get merged together is done by version control software. In this class, we will use the most popular and powerful version-control software available, i.e. Git and GitHub.

Git and GitHub

Git is a free and open-source version-control system designed to handle projects of all sizes with speed and efficiency. Users install Git onto their local machines (e.g., laptop) but unlike some software you might be familiar with, such as internet browsers, Git *per se* does not have a graphical interface. Therefore, we will

work with Git using the **shell**, which is a command-line way to access your computer's operating system's services.

GitHub is a web-based service for hosting projects that use the Git version control system. GitHub provides an attractive interface for viewing and managing a project's code, data, and text files. If your project is visible to others (public), then GitHub also serves as a way to let the world know about the awesome science you're doing and even how to join in and share tools. While many companies, agencies, and governments use GitHub (e.g., see https://government.github.com/), it is a great central location to manage any project.

Assignment 1: Create a GitHub account

- Navitate to https://github.com/ and click on the "sign up" icon
- Step 1: supply a username, your email address, and a password
- Step 2: choose the "Unlimited public repositories for free" plan
- Leave the "Help me set up an organization next" box unchecked for now
- Step 3: supply requested information to tailor your experience
- Make sure you bring your username and password to the first class meeting

Assignment 2: Read paper on Git and GitHub

- Blischak JD, Davenport ER, Wilson G (2016) A quick introduction to version control with Git and GitHub. PLoS Comput Biol 12(1): e1004668. doi:10.1371/journal.pcbi.1004668
- An open-access copy of the paper can be found on line, but is also posted on the course Canvas site: https://canvas.iu.edu

Assignment 3: UNIX tutorial As mentioned above, version control using Git and GitHub will require that you be competent with some basic UNIX commands. You may have been introduced to UNIX at some point, but it may have been years ago. That's OK. The following website will help you get up to speed http://www.ee.surrey.ac.uk/Teaching/Unix/

- Read "Introduction to the UNIX Operating System"
- Take "Tutorial One" and "Tutorial Two"

2) USING R, RSTUDIO, AND OTHER TOOLS

R is a programming language and software environment that can be used for performing statistics, graphics, and modeling. It is one of the most popular tools for conducting computational research in the life sciences, in part because it's free, open source, and has an active group of contributors. We are going to harness the power of R using **RStudio** (). Markdown is a simple formatting syntax for authoring HTML, PDF, and other documents. We will also use a tool called **knitr** (http://yihui.name/knitr/), which is a package that generates reports from R script and Markdown text. For example, when you click the "Knit PDF" button in the scripting window of RStudio, a document will be generated that includes LaTeX (http://www.latex-project.org/) typesetting as well as the output of any embedded R code. However, if there are errors in the Rcode in your Markdown document, you will not be able to knit a PDF file. All subsequence assignments in this class will require that you successfully create a Markdown-generated PDF using knitr; you will then **push** this document to the course **respository** hosted on GitHub (https://github.com/QuantitativeBiodiversity/QB-2017) and generate a **pull request**.

Introduction to RStudio

Eventually, we will make sure that your personal computer is set up with the proper versions of all necessary software. But for now, we are going to become familiar with a web-based version of RStudio that does not require an installations. Open a web browser and type the following into the address bar: https://rstudio.iu.edu. Now, we're going to do some simple exercises to get you familiar with how to use R and RStudio. There are many on-line resources (websites, videos, etc.) that can provide additional background and instruction. For example, here is a site that provides some information that will compliment some of the basics that we are about to explore: https://www.sitepoint.com/introduction-r-rstudio/

Executing code in Rstudio

There are a few different ways to execute (i.e., submit) your code in Rstudio. First, the lower left-hand panel of the Rstudio window is referred to as the *Console*. You can type your code directly at the **command line** in the console and hit enter. This is quick and easy, but it's not a great way to conduct reproducible science. A second way to execute code is to type it into the **script editor pane** in the upper-left panel of the Rstudio window. The code that you write can then be saved in an organized way as a *.Rmd file* when you are finished. You can execute code from the script editor pane by hitting Command+Enter (Mac) or Ctrl+Enter (PC). This will submit all of the code for the line where your cursor is located. Altenratively, you can highlight multiple lines of code and submit as described above.

Use R as a calculator

R is capable of performing various calcuations using simple operators and built-in **functions**. Use the **console pane** in the lower left of the Rstudio window to complete the following exercises:

Addition:

```
1 + 3
```

Subtraction:

```
3 - 1
```

Multiplication (with an exponent):

```
3 * 10^2
```

Division (using a built-in constant; pi):

```
10 / pi
```

Trigonometry with a simple built-in function (i.e., `sin`) that takes an **argument** (i.e., '4'):

```
sin(4)
```

Logarithms (another example of functions and arguments)

```
log10(100)  # log base 10
log(100)    # log base e "natural log"
```

Assigning variables in R

You will often find it useful and necessary to assign values to a **variable**, also known as an **object** in R. Generally speaking, in R, it's best to use `<-` rather than `=` as an assignment operator.

```
a <- 10
b <- a + 20
```

Check the value of b?

Now let's reassign a new value to a:

```
a <- 200
```

What is the value of b now? What's going on?

R held onto the original value of a that was used when assigning values to b. You can correct this using the `rm` function, which removes objects from your R **environment**.

```
rm("b")
```

What happens if we reassign b now?

```
b <- a + 20
```

Sometimes it's good practice to clear all variables from your R environment, especially if you've been working on multiple projects during the day. This can be done in a couple of ways. For example, you can just click `clear` in the **Environment tab** of the **Workspace and History pane** in the upper-right paenel of the R Studio window. The same procedure can be performed at the command line of the console or script editor pane. To do this, you can use the `ls` function to view a list of all the objects in the R environment:

```
ls()
```

```
## [1] "a" "b"
```

You can now clear all of the stored variables from R's memory using two functions: `rm` and `ls`.

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

Working with vectors in R

Vectors are the fundamental data type in R. Often, vectors are just a collection of data of a similar type, either numeric (e.g., 17.5), integer (e.g., 2), or character (e.g., "low", "medium", "high"). The simplest type of vector is a single value, sometimes referred to as a **scalar** in other programming languages. Again, use the **console pane** in the lower left to complete the following exercises:

```
w <- 5
```

We can create longer one-dimensional vectors in R like this:

```
x <- c(2, 3, 6, w, w + 7, 12, 14)
```

What is the function `c()` that we just used to create a vector? To answer this question, type `help()` function at the command line in the console pane.

```
help(c)
```

You can also retrieve the same information using a question mark followed by the keyword of interest

```
? c
```

What happens when you multiply a vector by a "scalar"?

```
y <- w * x
```

What happens when you multiply two vectors of the same length?

```
z <- x * y
```

You may need to reference a specific **element** in a vector. We will do this using the square brackets. In this case, the number inside of the square brackets tells R that we want to call the ith element of vector `z`:

```
z[2]
```

```
## [1] 45
```

You can also reference **multiple elements** in a vector using the square brackets and the colon symbol:

```
z[2:5]
```

```
## [1]  45 180 125 720
```

In some instances, you may want to change the value of an element in a vector. Here's how you can substitute a new value for the second element of `z`:

```
z[2] <- 583
```

Summary Statictics of Vectors

It's pretty easy to perform summary statistics on a vector of data using the built-in fuctions of R:

```
max(z)    # maximum
min(z)    # minimum
sum(z)    # sum
mean(z)   # mean
median(z) # median
var(z)    # variance
sd(z)     # standard deviation
```

What happens when you take the standard error of the mean (`sem`) of `z`?

The standard error of the mean is defined as $\frac{sd(x)}{\sqrt{n}}$. This function does not exist in the **base package** of R. Therefore, you need to write your own function, either in the console or the script editor panel Let's give it a try:

```
sem <- function(x){
  sd(x)/sqrt(length(x))
  }
```

Actually, there are a few functions inside of `sem`. Take a moment to think about what is going on here. Now, use the `sem` function you just created on the vector `y` from above

Often, datasets have missing values (designated as 'NA' in R):

```
i <- c(2, 3, 9, NA, 120, 33, 7, 44.5)
```

What happens when you apply your `sem` function to vector `i`? This is a problem! We can remedy the problem with the following modification of our `sem` function.

```
sem <- function(x){
  sd(na.omit(x))/sqrt(length(na.omit(x)))
  }
```

Now run `sem` on the vector `i`. What did we do?