**SSI Coding Notes & Version Control SOP**: *On Git, R, and the Shellements of Style*

**

This SOP will cover what GitHub and Git are and the workflow we have setup at SSI -so far- for getting started with coding and Git. This SOP is meant to help expedite setup and using our GitHub, a quick and dirty intro to R and R Studio, as well as set some basic coding standards as our organization moves more and more into the coding world. The standards aspect is included in the hopes our GitHub does not go the way of our server- disorganization to the point of uselessness- as well as to set code standards that account for our organizations’ high turnover and need for accessibility- so our scripts are friendly to staff of all familiarity levels. This tutorial may hop-scotch because all these things are connected, but here is a Table of Contents below to help find what you’re looking for in a pinch.

Otherwise, this SOP will start with a general intro to Git, Github and our repository, then do a quick example of writing an R script, cloning a repository, and pushing files and changes to your GitHub repository using Git.

Anyway, let’s start.

*Note: I wrote this SOP as if the reader has never used any of the software or concepts before, or has used them very minimally. That was my experience, and in a way, I’m directing this SOP to my past self and what would’ve fast-tracked my process a year ago- going from no-coding to coding a lot. Sorry for the step-by-step for any Wiz’s that are looking for a tidbit of information in the haystack. And, of course, all readers should edit and add as you see fit!*

*Table of Contents:*

1. *Git and GitHub Introduction and Vocabulary*
2. *Very quick example of how to use R, Git, GitHub*
3. *Required Installations and Basic R tutorial*
4. *Basic Git and GitHub tutorial*

*------------------------------------------------------------------------------------------------------------------------------------------*

*1) Git and GitHub Introduction and Vocabulary*

GitHub is a code hosting and collaboration website. Its primary purpose is allowing a user, or multiple users, to write and edit code (and other files) from their local computers, and then update their local work to and from an online central project files location. The central project, where the shared version of the code lives, is called a **repository**- and that’s the set of files on GitHub. The version of the project on your local computer is called a **clone.** When you make edits to your local code (or other files in the repository, such as CSVs or .txt files) and you want to update the shared code in the repository with your new edits, that is called “**pushing**.” When you want to make sure your local files are the latest version you can update your local files to the latest version of the code and associated files on your repository by a process called **pulling** (or **fetching**). These terms will become clearer as we go through the tutorial later in the SOP. There is also very good **documentation** on the internet of all of these and more about [getting started](https://docs.github.com/en/github/getting-started-with-github) with **git** and **GitHub**.

This whole process- of sharing, pushing, and pulling, is called **version control.** While at first GitHub can seem like its function is basically like a google drive for code, its real purpose is to clarify and ease the control of information and updates in a multi-collaborator project. The reason why this distinction is helpful and important will be discussed more later in the SOP, but as you start to manage projects in GitHub you will see how much time and anxiety is saved by sharing files and updates this way, and not through enormous email chains with a bunch of file names like:

*“finalUPDATE\_version2\_WS\_DP\_FINALVERSION\_Update3\_06\_6\_2021.R”*

Before we dive in it may help to clarify: **Git** is a programming language that is executed through a **Bash** [**shell**](https://en.wikipedia.org/wiki/Shell_(computing))**.** This is complicated enough without going into shells, but in short, a **shell** is an interface where you can “talk to” your operating system directly. I think about it as “talking to,” which is in not an accurate technical description, but you know, you say potato, I anthropomorphize technology. **Git** looks like this:

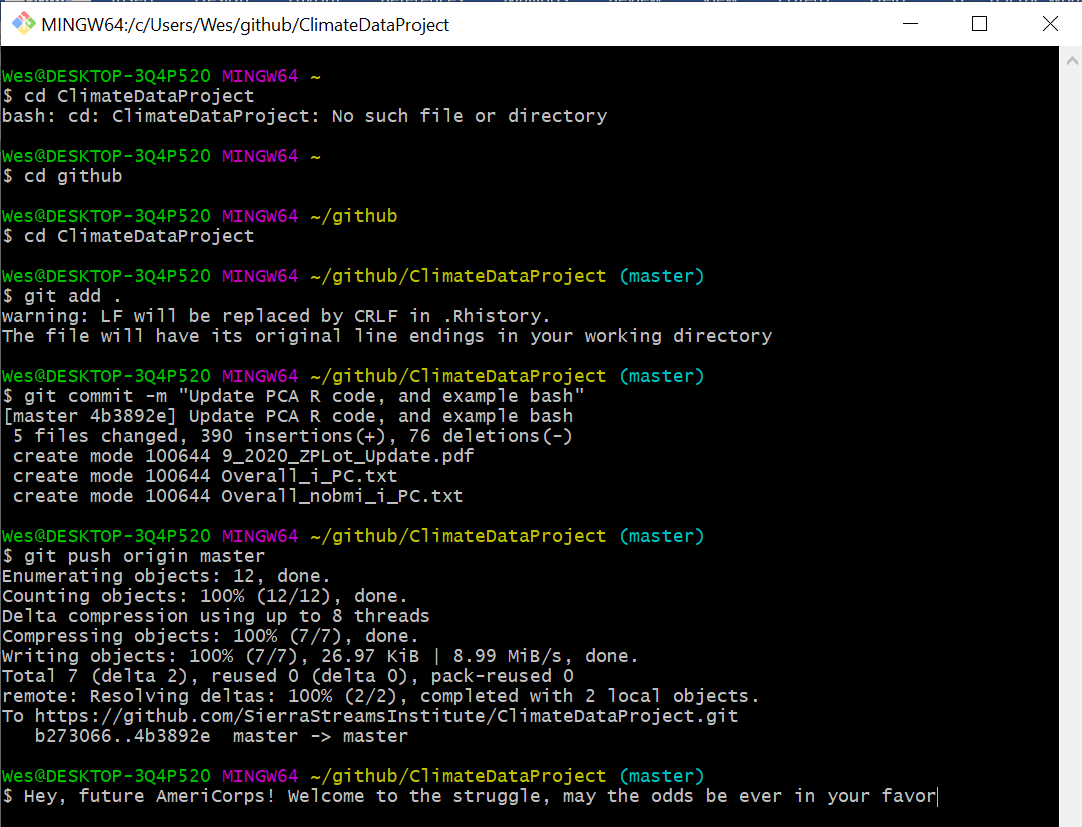


Figure 1 Git in the Bash Shell

Believe it or not- you will understand every line of code typed in *Fig 1* by the end of this SOP! Its bark is *way* worse than its bite.

**GitHub**,on the other hand,is a website that facilitates the user-friendly use of **git** to manage projects.

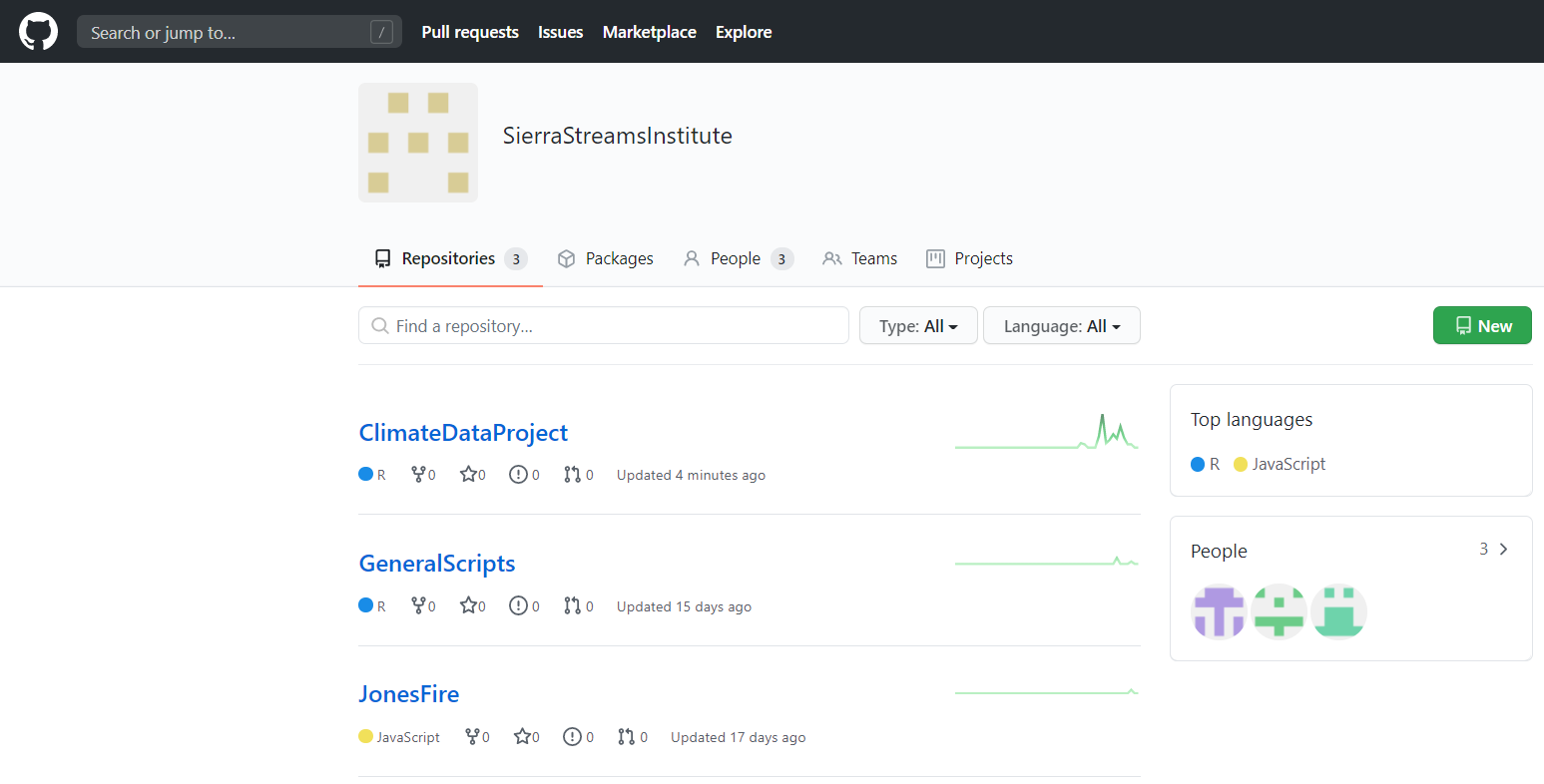
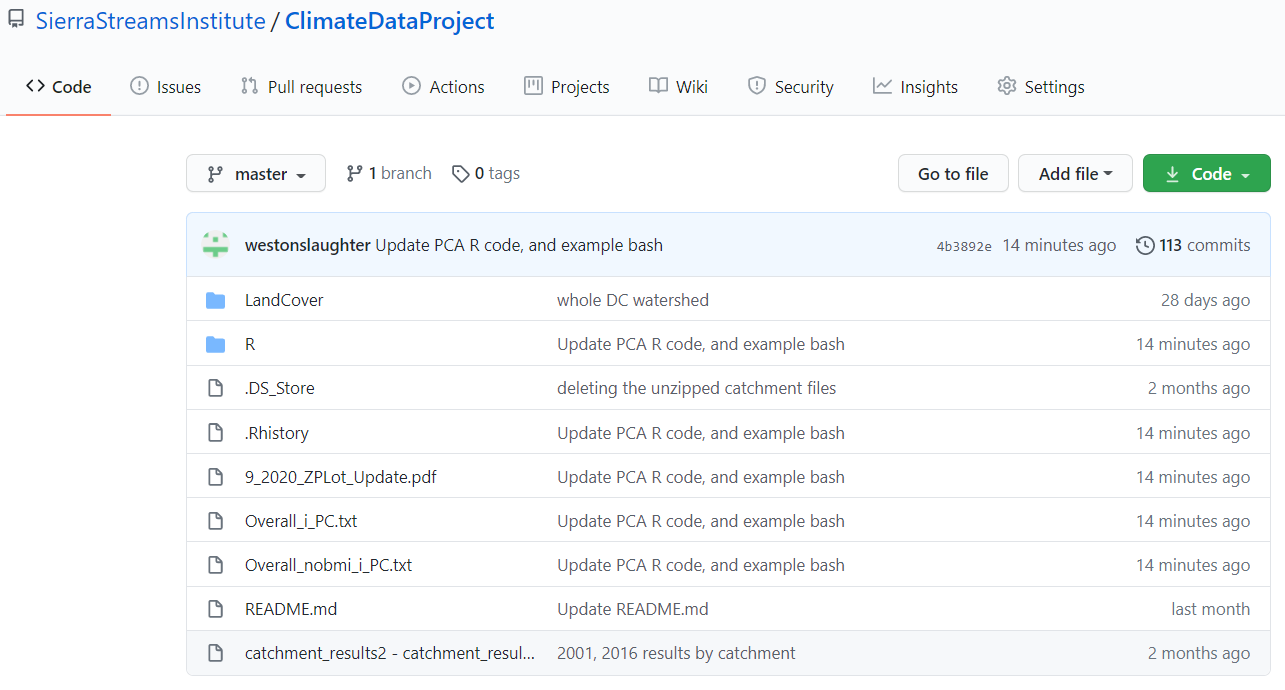
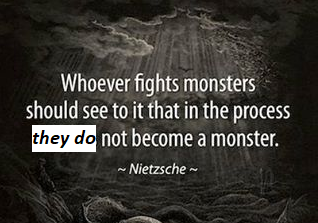


Figure 2 SSI GitHub page circa September 2020

What’s that you say? “That looks more inviting than the **git bash**thing” yes, exactly! You can see we have three projects at this time, each with its title and some neat info describing what’s inside. let’s step inside our most active one, the **ClimateDataProject**:



Oh no, you say, you told me this was *better* than the server, but that looks like horrible file organization! You’ve simply created the monster you were fleeing in the first place! Yes, yes, that’s true. A mantra we have failed to live by:



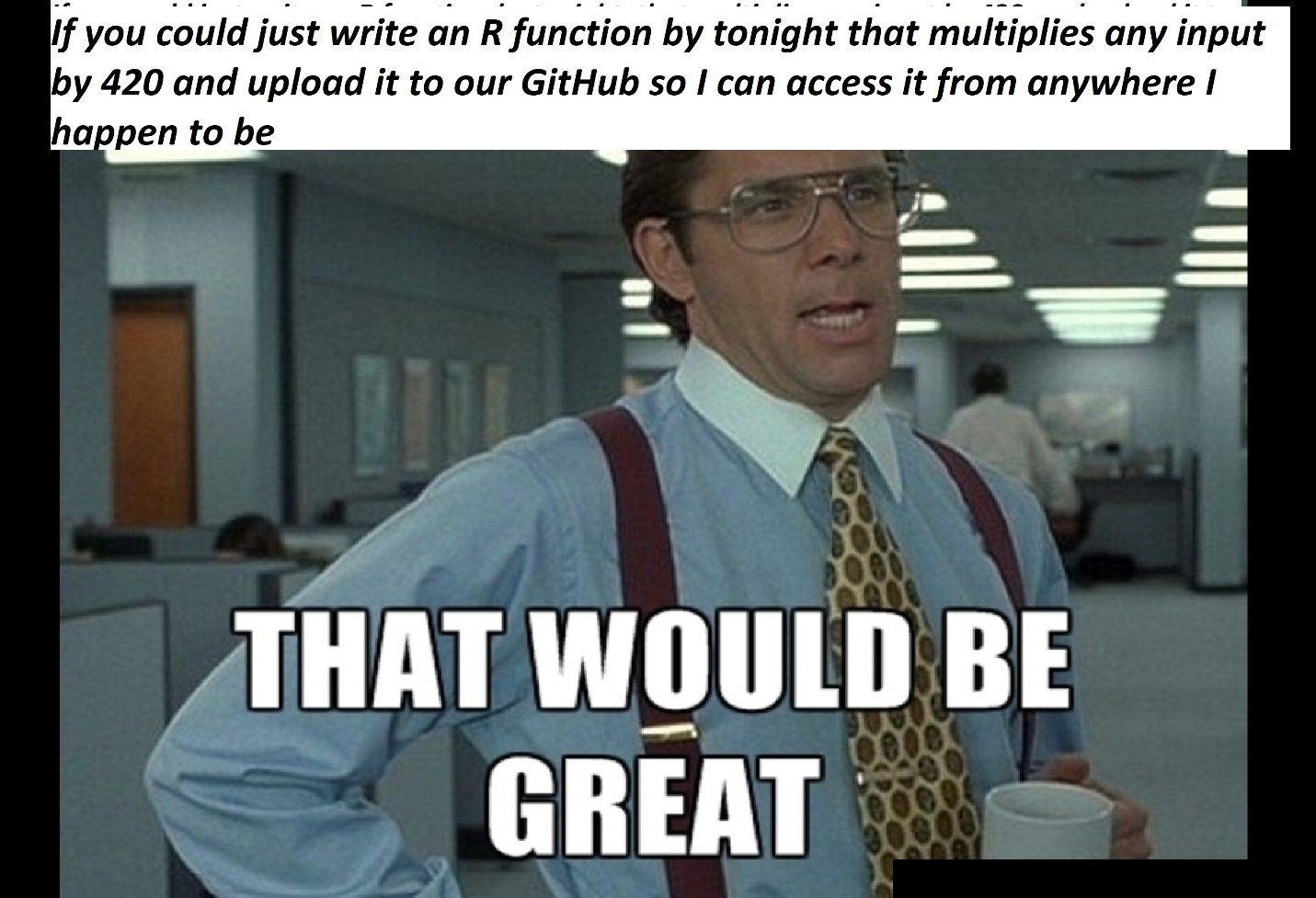
Alas, nipping the already head-rearing disorganization monster in the bud is a duty that has fallen into your capable hands. We’re sorry.

Now we’ve seen the language Git as it looks in action, and we’ve seen our GitHub page, and looked into the files of a project. But that’s enough small talk, let’s do it live- in Part 2: Quick Example of how to Use R, Git, and GitHub

2) Quick Example of how to Use R, Git, and GitHub

1. *Required Installations and Basic R tutorial*

So, let’s say it’s your first week, and it’s around 6pm and you’ve just finished up getting assigned 20 projects from 7 different supervisors, when Kyle Leach pops into your office and says:



After the panic settles, you hit this SOP like a meteor, and read that you need three things:

1. R and R Studio
2. Git
3. Your wits about you

*Solution to thing 1: How to Download R and R Studio*

This is a cop out, but this Medium article explains it better and faster than I ever could: [Downloading R](https://towardsdatascience.com/how-to-install-r-and-rstudio-584eeefb1a41)

*Solution to thing 2: How to Download Git*

Again a big cop out, same reasons, a great [Git install guide](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)

*Application of thing 3: How to Use R and Git to Make Friends and Influence People*

You’ve got R and R Studio, and you’ve got Git. Where do you go from here? Well let’s start with writing your code. *\*ASIDE\* This SOP is not intended to overview how to code, but it will go over styling and standards for code. At some points, the line blurs between the two, and I’m just going to lean into that \*ASIDE OVER\**

So open up R Studio and you’ve got this whole situation:

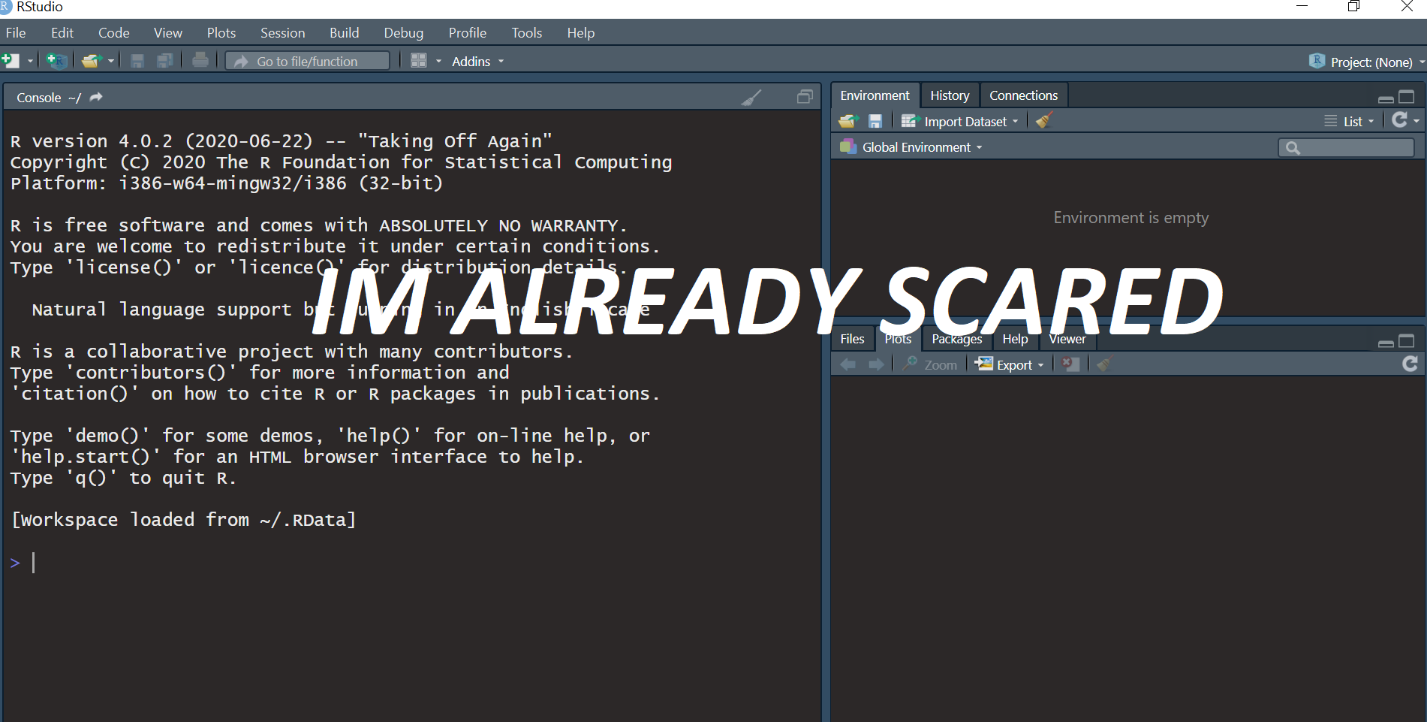


Figure 3 R Studio blank situation

But don’t fear. Navigate to **File** in the upper left corner, and then do **new file > R Script**. I spent the first two weeks in R markdown, and it was a scary, unforgiving place.

*We’re about to write our first R Script! Before we start, it’s worth saying- actually code along with this script, or at least write something. You’ll need an R file on your computer for the Git tutorial section.*

So now you’re on your first R Script, but before you start the actual code, it’s important to record, at the top: **the title, author, organization,** and **date.** You can write directly in the script file like any text document. In a script you will have **code** and **comments.** Code is anything that the computer will read when you **execute** a line of code or a script. Comments are notes you put in to help yourself or other readers to understand what’s going on in your script. Anything you type in is automatically code- but you can turn code into comments by typing **Ctrl+Shift+C** on the line you want to “comment out” or by doing **Ctl+Shift+C** before you start typing. You can also manually put the “#” sign before you type, which will also designate the line as comment and not code. You can **execute** or **run** a line of code (make the magic happen) by typing **Ctl+Enter** to run the line your cursor is on OR type **Ctl+Shift+Enter** to run all the code in the script. Here’s an example of the script that Kyle was asking for:

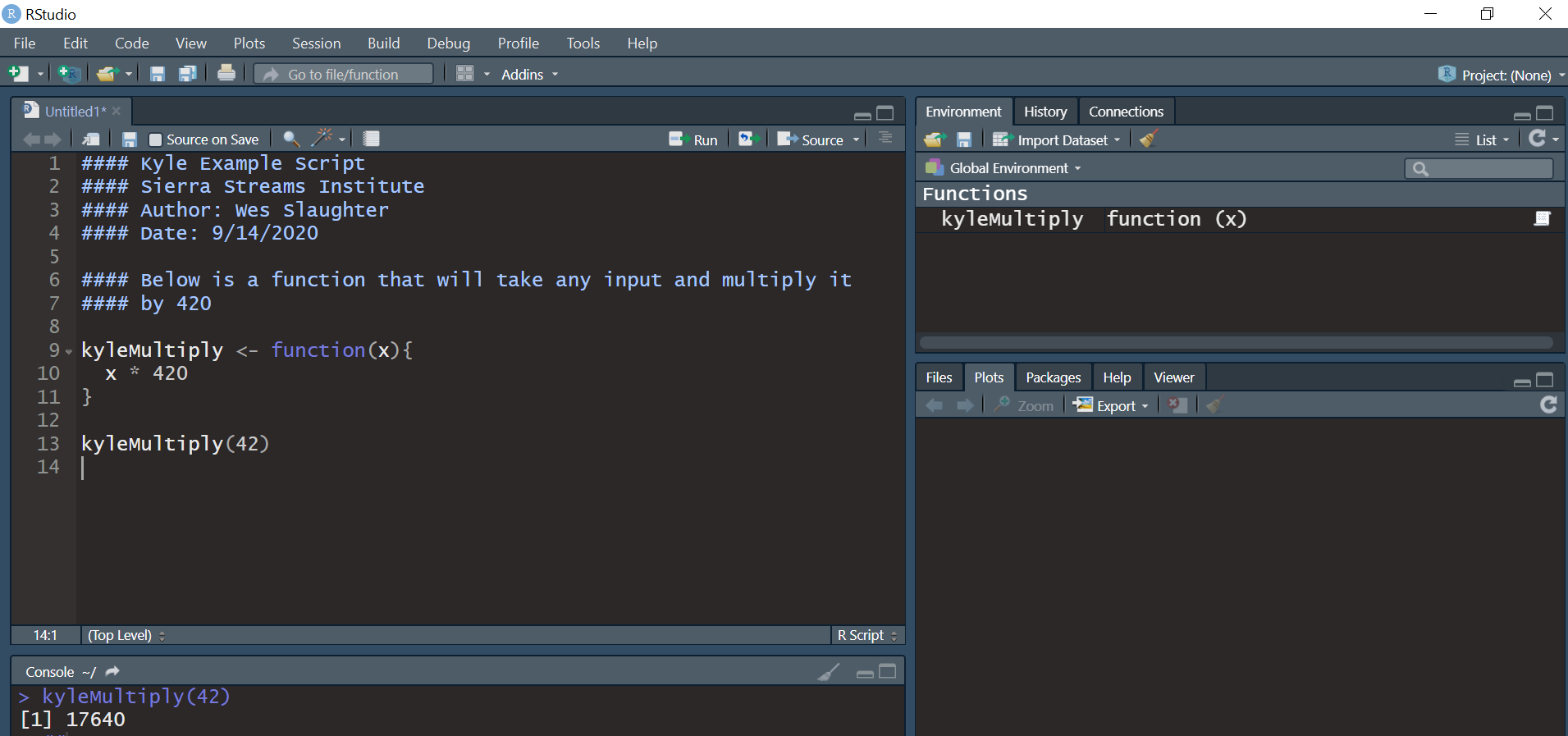


Figure example of a script in R with conventions

*Crash Course Coding How-To’s and Conventions*

**Lines 1-4:** These lines are “comments” because they have **#** signs in front. Whatever the color scheme of your R Studio interface, it will likely change color, as mine does. I pressed enter to put a spacing line before my next comments.

**Lines 6-7:** More comments, this time, to describe what the code below contains and what it does. Again, pressing enter to make space to improve readability.

**Lines 9-11:** A lot here. So, this is our **code**. The first important thing here is the **<-** symbol. This is called the assignment operator, and is the primary means by which you will define objects, variables and functions. For example, if I wrote:

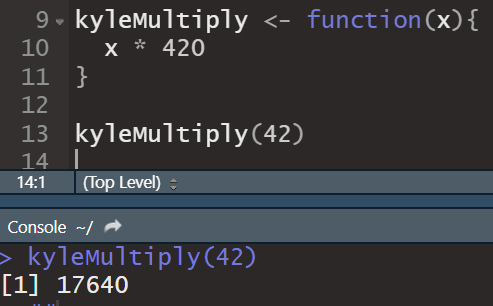
*exampleVar <- 4*

The computer will recognize the typed word “exampleVar” as the numerical value 4, and you could use it in calculations or any context as such. Learn more about operators [here](https://www.datamentor.io/r-programming/operator/) (assignment operators are covered at the bottom of the webpage, for some strange reason).

In this case, I am using the assignment operator **<-** to assign a **function** to the text “kyleMultiply.” Now, the topic of creating your own functions may be a little too advanced for our purposes (at least it took me a while to wrap my head around it), so [here](https://www.tutorialspoint.com/r/r_functions.htm#:~:text=User%2Ddefined%20Function,squares%20of%20numbers%20in%20sequence.) is a resource to learn more about R functions and user-defined functions. In short, for any function you write

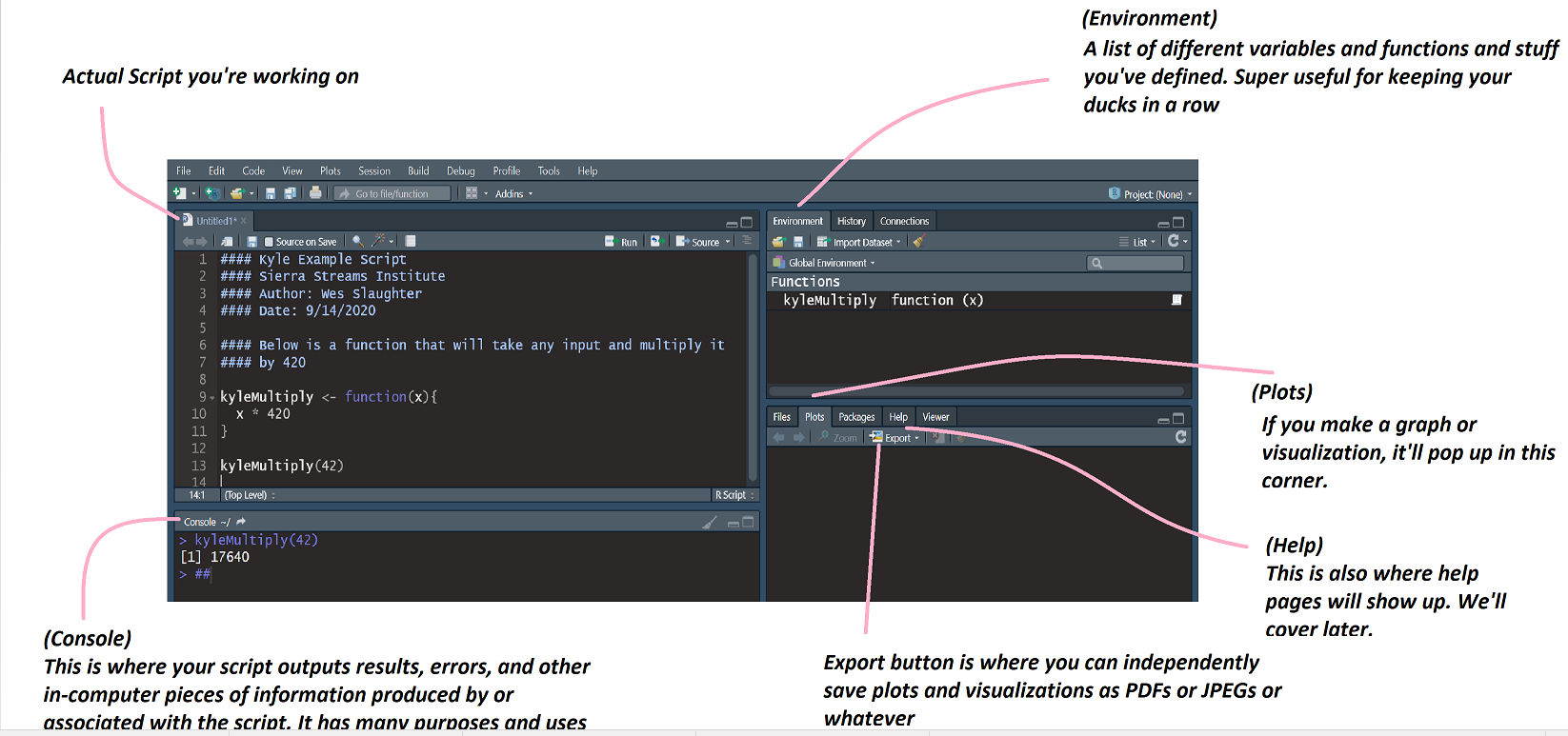
*function(){}*

after the assignment operator. Within the normal parenthesis (), you type in one or more variables. In the curly brackets {}, you type in what your function will automatically do to your variables, such as multiplying by 420. Once defined, you can use your function with real inputs, and it will apply whatever operations you dictated in the curly brackets to that input.

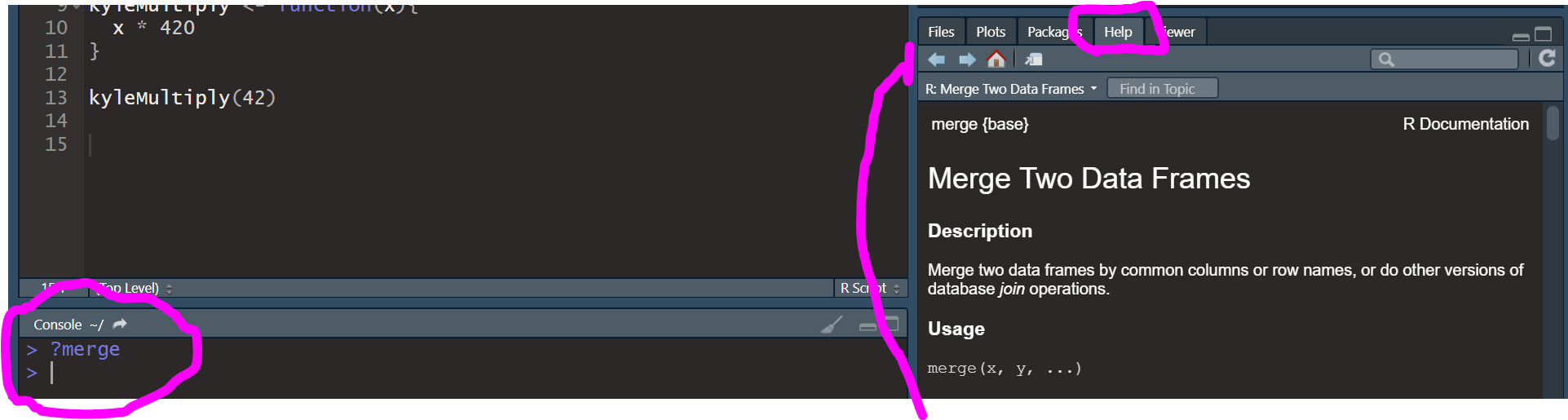


If you feel overwhelmed that’s totally okay. It took me forever combing thru stack exchange and practicing and re-learning over and over again to feel comfortable with even the simplest of the topics we just breezed through. Mostly, I don’t want to pass over a detail of this SOP without at least calling out what’s happening, and providing resources. For me, one of my biggest obstacles was coding tutorials and resources that assumed I had basic knowledge I didn’t, and it could make it feel impenetrable.

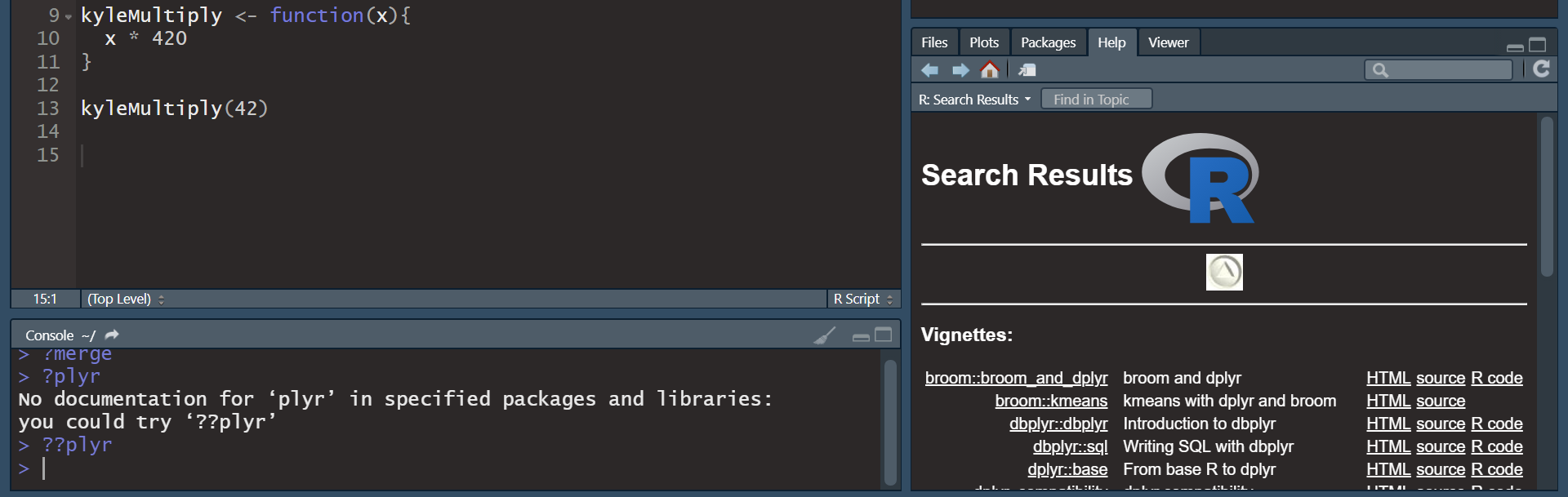
R Console Layout Rough Guide:



**HINT** (help pages): if you type **?** before the name of something in the console, and run it, a “documentation” will pop up about it, if documentation is available.



Sometimes, there is no out-of-the-box documentation for something in R, but there is officially-sanctioned documentation online. You can search thru this larger online help catalogue with **??** here is an example:

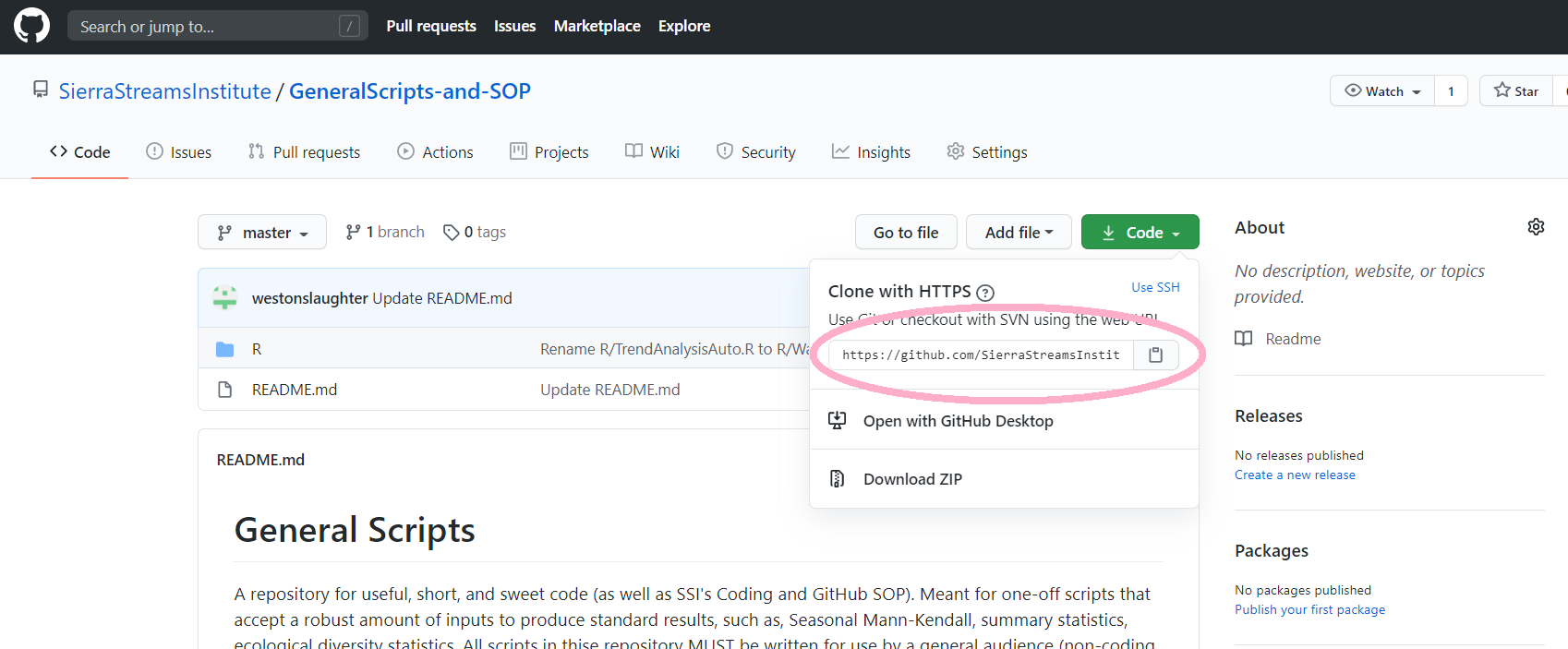


Though, in general, if there’s no help from **?** results, I usually find googling it is more useful than using **??** to each their own tho

1. *Basic Git and GitHub Tutorial*

So you’ve got your script, but now you need to put it on GitHub.

First, go to [SSI's GitHub page](https://github.com/SierraStreamsInstitute). You figure your multiplication script is pretty darn useful, so let’s open up the “ScriptGeneral” repository. You know this is where you want to put your files. First, you need to **clone** the repository to your local computer. To do this, click the green “**Code”** button on the right. Then, copy the link in the box that pops up.



Okay. Now, we dive into Git. Go to your computers search bar (Finder, or the start menu search bar), and type in Git. If your download was successful, something like this will show up.

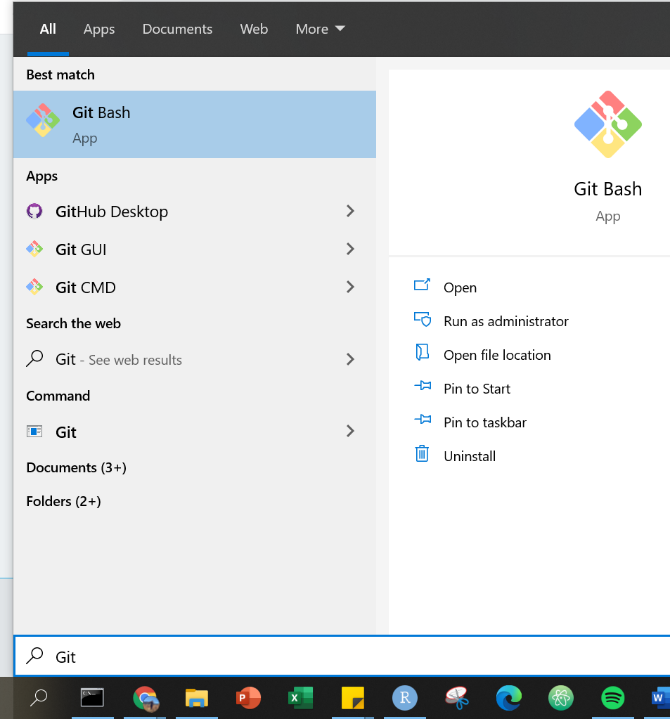
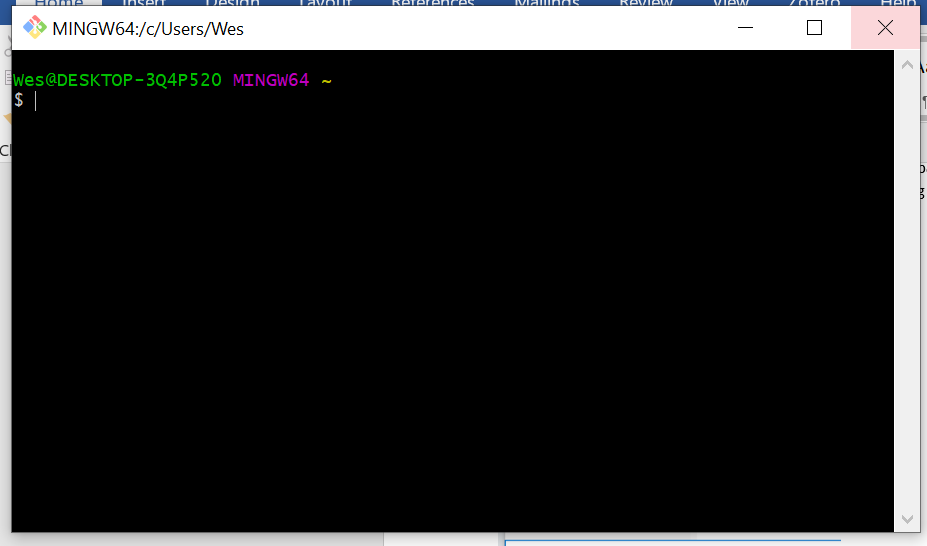


Figure honestly, go ahead and pin Git Bash to your taskbar like a boss

Click on Git Bash, and this should pop up.



Here we go! So, first step- where do you want to have this repository on your computer? The text to the above-left of the cursor may look confusing, but the translation is just that this console is located at my Desktop’s “Wes” folder, so basically like your normal file explorer location terminology. To visualize this:

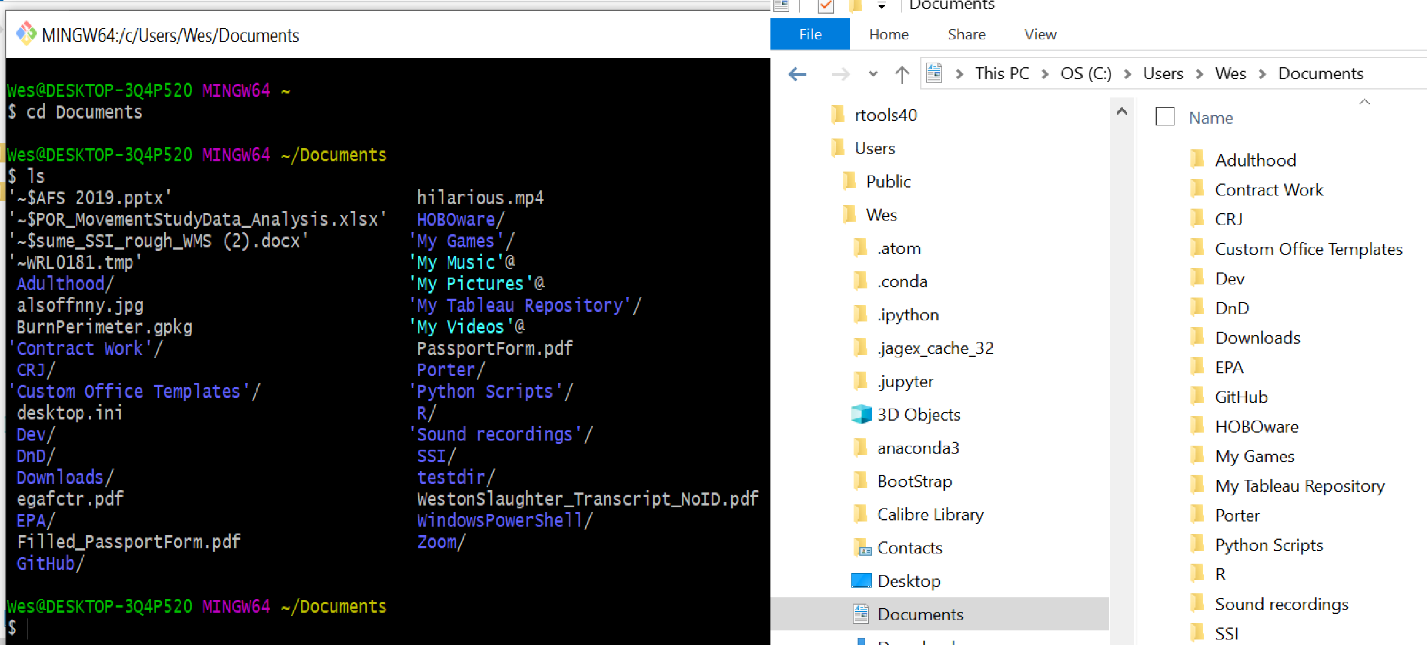


Figure Git Bash vs File Explorer comparison. For reference "ls" is a command that will show the contents of the directory at which the console is. This can be helpful to orient yourself to know what directory you may be in, or verify you're in the correct one. If you look at the files listed on the bash console (left) and the file explorer (right) you can see they are showing the same file folder, just very differently. The bash console is closer to how your computer “sees” it.

So, you’re starting out by navigating your computer’s files in the console instead of by pointing and clicking in file explorer. You can move forward by typing “**cd \_\_\_\_\_\_\_**” with the “\_\_\_\_\_\_” being the exact name of the sub-folder you want to navigate to. You press enter to run a command. You can move backwards in the directory by typing “**cd ..**”

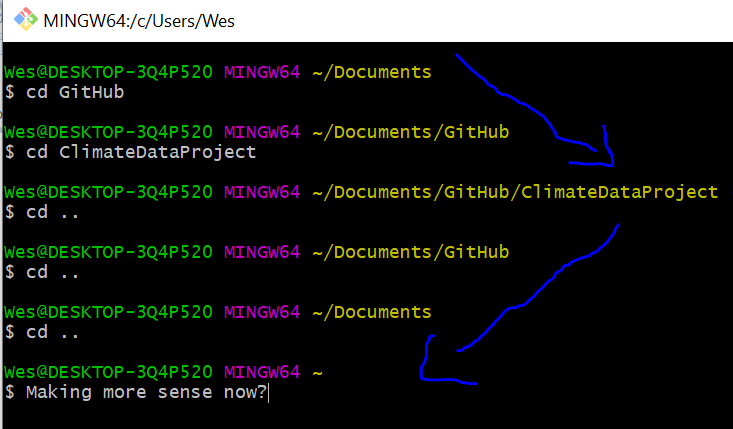


Figure Feel the exhilarating power of the console!

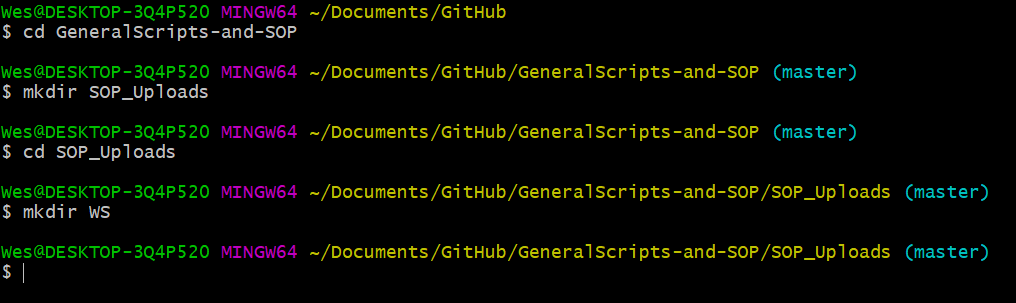
So, anyway, in my case, I want to clone the repository in my “GitHub” folder. I use the **cd** command to navigate to the folder I want to use on my local computer. Then to clone the repository in simply type into the bash console “**git clone *[paste the URL you copied from the GitHub repository]***” the console should then print out the status reports on the cloning process, as it appears below, ending with several **done**’s.



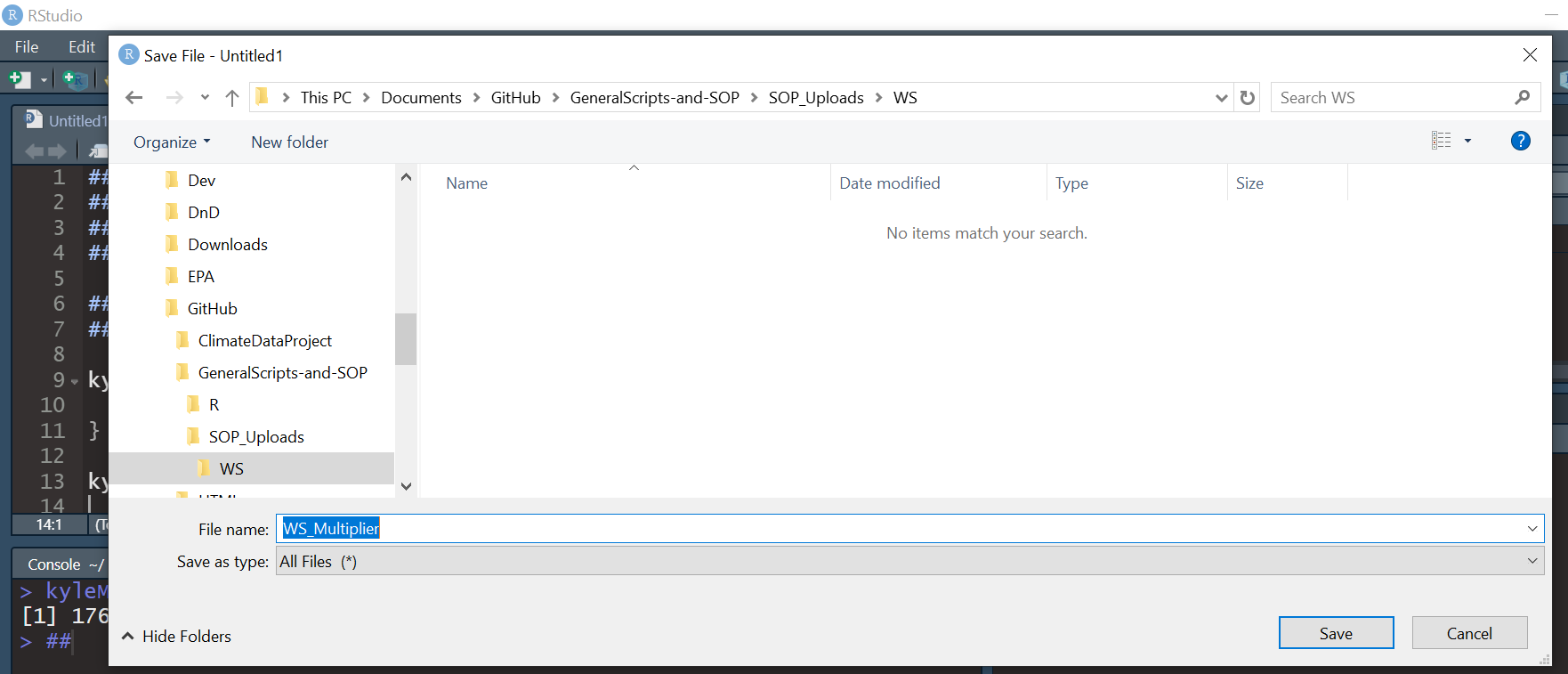
Figure Even MORE extreme power! Muahaha

Now, a folder named **GeneralScripts-and-SOP** should be in your folder if you look on File Explorer. For me, this is the part of the process where I realized how the Bash console, though intimidating, is actually extremely powerful and can execute complex tasks very quickly and simply. In any case, it’s time to upload your script.

For this SOP, every person completing this tutorial will make a folder with their initials to upload their own “kyleMultiply” script to. To achieve this, you will use a command **mkdir**. In my console screenshot below, you can see me using **mkdir** to create a folder called “**SOP\_Uploads**” and then I navigate into this folder, and create a folder called “WS.” Do this with your own initials by navigating to the **SOP\_Uploads** folder, and typing in **mkdir [*insert your initials]*** (my initials are WS)



Now that you’ve made your folders, you need to save your script into the folder you just made and titled with your initials. There’s probably a nifty way to do this in the Bash console, but I would just go to R, and “Save As” your R script into the folder you just made.



Now you have your local clone of the repository, and you’ve updated it with your personal folder and the script you made. Now, it’s time to **push** the changes you’ve just made to SSI’s GitHub, so that Kyle, or any of the staff, can access it from anywhere. This also happens in a couple lines of code in the console. Git needs you to **stage** your **commits** before you can **push.** This lingo can be confusing, and admittedly, I don’t completely understand what it’s all about. However, basically, you can’t just go straight to **push**ing your changes- Git needs you to tell it what of your local changes you want to make happen on the shared repository. Staging is a system of collecting the changes you want to make official. Committing is what it sounds like- re-confirming you want to make those changes. Then, once you’ve done that, you can **push-** and the changes will be made. I will show you how to just push all of your local changes. This is what I always do because I’m not good enough at Git to understand the more complex picking and choosing changes process.

So, to commit all your changes- type

**git add .**

and press enter. This stages all of your changes (the **.** symbol means “all the changes I’ve made”)

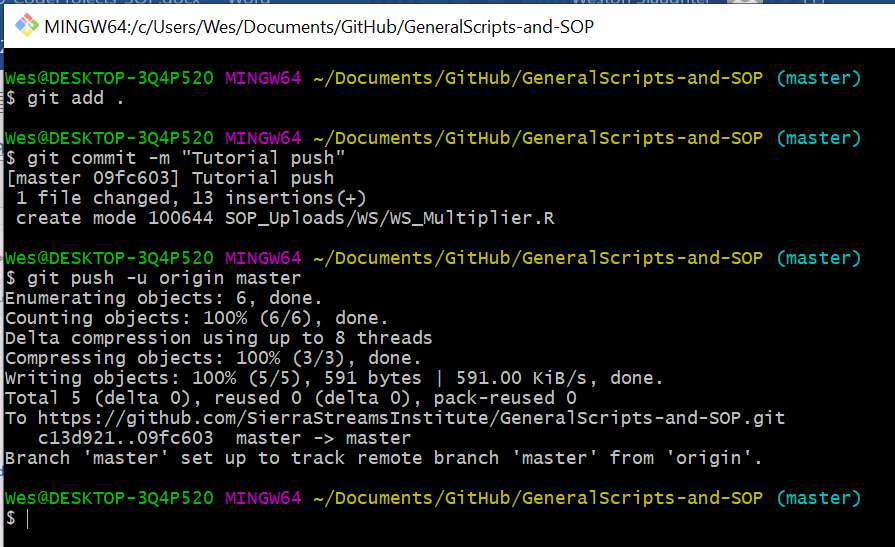
then type

**git commit -m “[*type any message here, it should be a description of what the changes you’re making are. This message appears in GitHub and can be very useful for your team*]”**

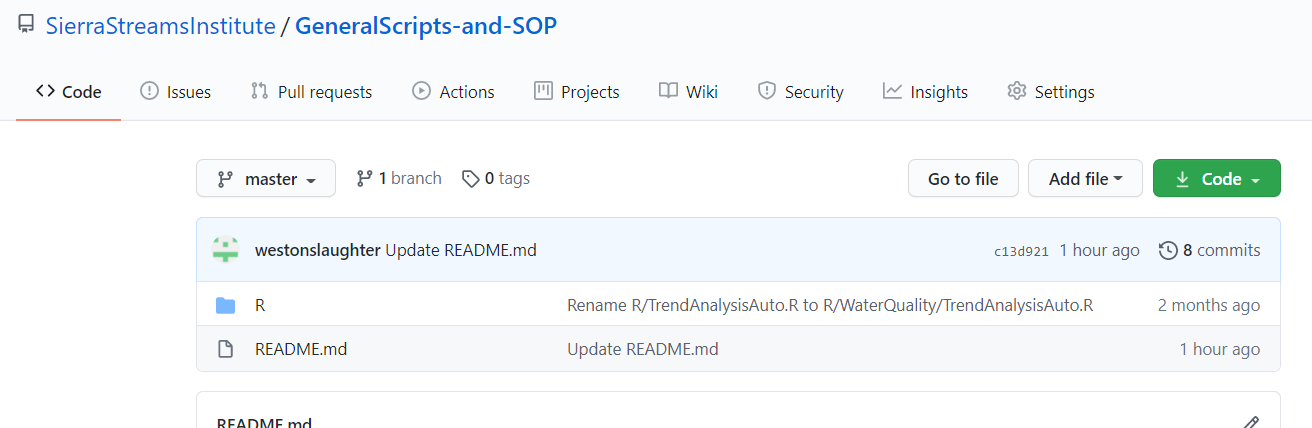
press enter again. Now type:

**git push -u origin master**

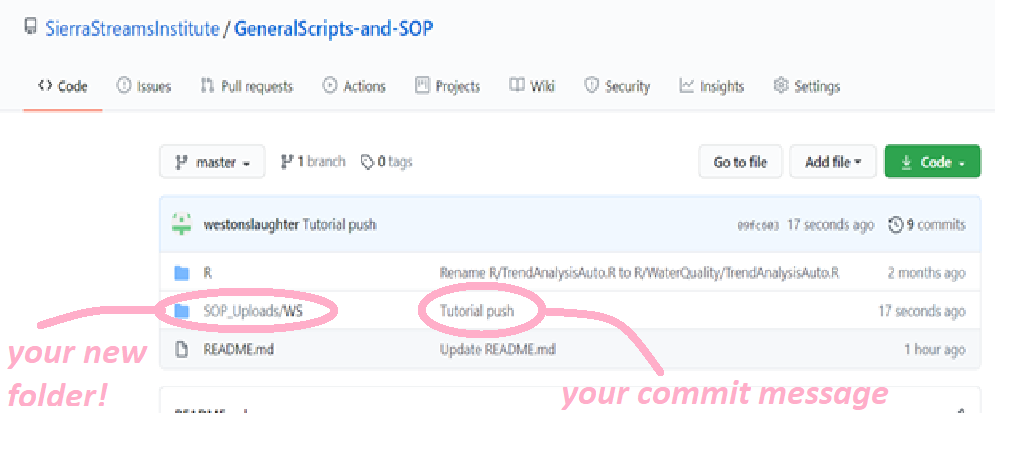
press enter, and the consoled reads out a report on your upload and changes.



To illustrate what you’ve done, this is the projects GitHub page before those commands:



And here’s after:



And your codes up now! Congratulations, you can apply this workflow to make folders, upload and update scripts of any kind on our GitHub!

