

Getting familiar with RStudio

RStudio is a free and open source Integrated Development Environment (IDE) that makes developing R scripts easier.

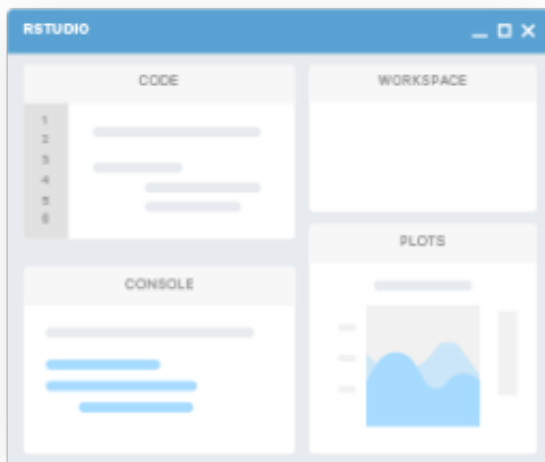
In addition to the R console, this software provides: a script editor, a visual capture of your commands history, the contents of your environment, plots history, file browser, interactive debugger, and more. With RStudio, you can also manage code versioning and publish documents.

Download RStudio here: <https://www.rstudio.com/products/rstudio/download3/>

We recommend that you install RStudio Desktop (free license) on your personal computer. RStudio server can also be installed on a server to allow access to the RStudio IDE via your web browser (See the RStudio Server Section below for more information).

Navigating inside RStudio

The RStudio environment consists of four main panels: *Code, Console, Workspace* and *Plotting area*



(image source: <https://www.rstudio.com>)

The **Code Panel** is where you can view, edit and execute/source R scripts. Technically, any piece of code run in the code panel is no different than entering the code into the console, but the code panel makes it easier to view and edit code by providing syntax highlighting and indentation to help you to organize your script. You can also execute code line by line in a reproducible manner. To run chunks of code in the code panel, highlight lines and click the "Run" button (upper-right of code panel) or press `Ctrl` or `Command` on a Mac and `Enter`. You can easily copy and paste text from other places in the code panel, or copy and paste into the console. Finally you can execute the entire script by clicking on the "Source" button.

The **Console** is where you can type code manually, or paste code from the code panel or other programs. If you have used R without RStudio previously, this console is basically the same as the classic R console. Running more than a few lines of code in the console can be clunky, especially when you need to troubleshoot errors, so we recommend using the console for 1) testing small pieces of code or basic functions, 2) inspecting the output of your code, 3) viewing error messages.

Note: your current working directory is displayed at the top of the console. You can manually set a working directory at any time, but the default directory is used when a new session is started. The default can be changed under *Tools -> Global Options -> General*.

The **Workspace Panel** by default displays your R global environment, which contains the names of the variables and datasets your code defines (e.g., data frames, matrices, strings, etc.). Some datasets can be viewed by double-clicking the name in the environment. The "History" tab shows what code has been previously run in your session. The same history can be viewed in less abbreviated form (errors and printed output included) by scrolling through the console. You can select lines from the History and use the "To Console" or "To Source" to send automatically the selected commands to the R Console or to your R Script (where the cursor is located), respectively.

Despite its name, the **Plotting Area Panel** contains several important tabs. One of which is "Plots" that displays visual results of R code (e.g., graphs, maps) and allows you to browse your plots history. Note the "Zoom" and "Export" buttons for exploring and saving your visualizations. The "Files" tab displays your directory structure (based on your default or defined working directory). The "Packages" tab contains all the loaded (checked) and installed but not loaded for your current session (listed, but unchecked) R packages in your System Library. You can install a Package by clicking on the "Install" button. Clicking a package name brings up its documentation in the "Help" tab. You can manually navigate or search for documentation in the "Help" tab, but also note that typing `?function_name` in the console (e.g., `?cbind`) locates documentation for a particular function in the "Help" tab. Functions from uninstalled packages cannot be found in the "Help" tab until respective packages are installed. The ["Viewer" tab](#) is for local web content, often from specific packages.

Note that the organization and size of the various panels, as well as the contained tabs, can be customized by the user.

RStudio Projects

This section is adapted from the more detailed material available on the RStudio website:

<https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects>

RStudio Projects enable you to organize your work by setting a directory on your machine as the main directory. Note that by default, this directory will be set as the working directory. To create a new project you can use the menu *File -> New Project* or the drop-down menu from the top right corner of the RStudio IDE. From the top right corner you can also see the recently opened projects. When you create a RStudio Project, a file with the extension `.Rproj` will be created in the main directory. Once you are working within a project this same top right area will display the name of the project you are currently working on. By default, a Project will save the current status of your session: workspace (including data loaded), history and opened scripts. When you reopen a project, all of these configurations will be loaded for you, bringing your session back to the exact state you left it.

Note that to use code versioning through the RStudio IDE, you need to create a RStudio project.

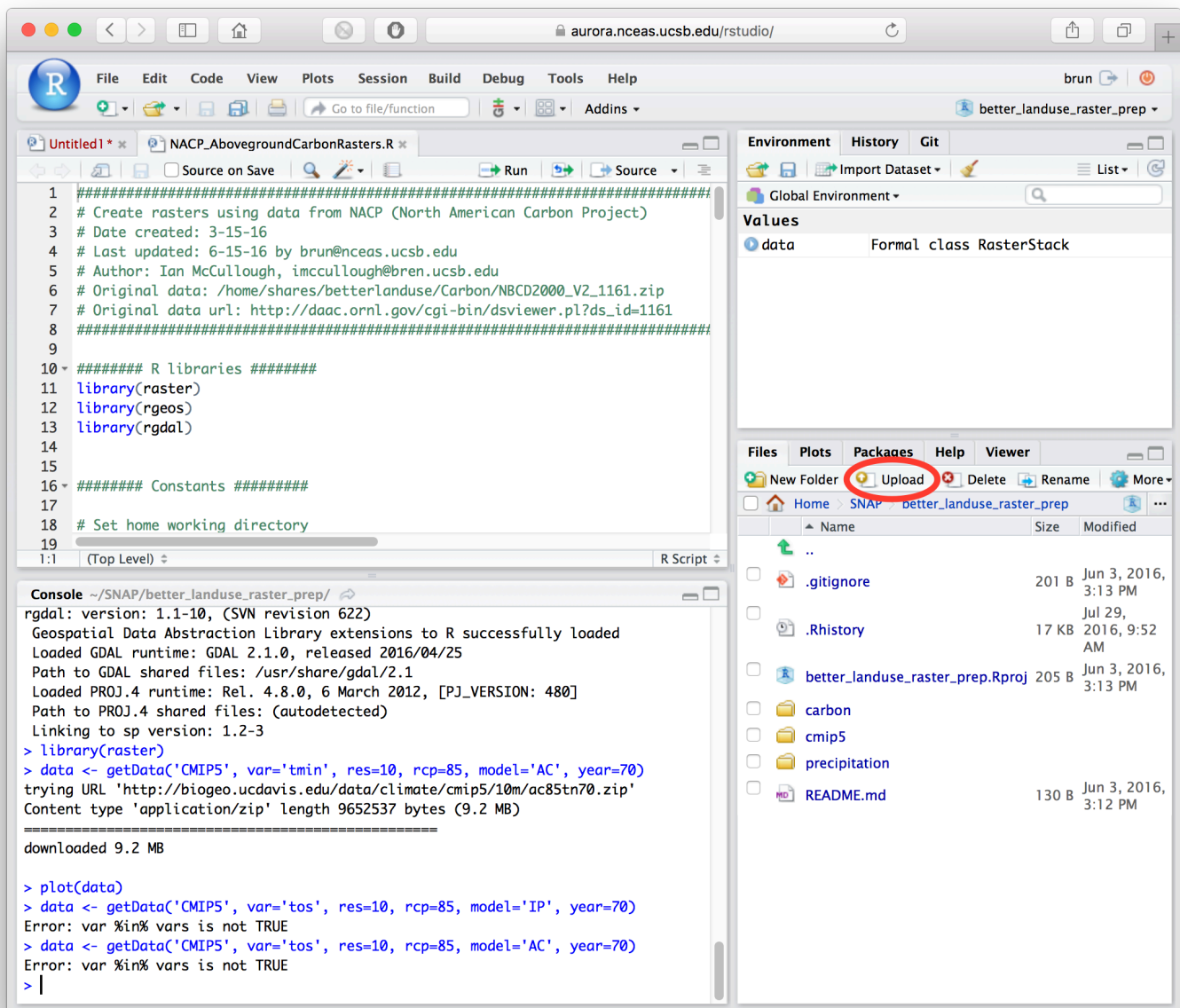
However, we do not recommend you track the `.Rproj`, `.RData` and `.Rhistory` and other `.R*` files present in the main directory. See the Section on code versioning below for more information.

RStudio Server

RStudio server is simply RStudio running on a remote machine that you access via your web browser. For example, **RStudio server is running on the NCEAS analytical server Aurora**. You can connect here: <https://aurora.nceas.ucsb.edu/rstudio>. If you do not have an account on this server, please coordinate with your Working Group leaders to obtain one.

Notes there are 2 main differences:

1. With RStudio Server you are running your job on a remote machine, possibly leveraging the large amount of RAM and CPUs available on that host compared to your desktop/laptop. Also, when you close your browser window, you do not stop the process/script running! It can be very useful to disconnect and reconnect later, as script execution continues and your code and history is just like you left it.
2. You will need to move your files and scripts to the remote machine in order to access them. Note the extra upload button on the files tab of the plotting area (circled in red in Figure below) that allows you to transfer one file at a time. If you have several files to transfer, you can either create a zip file or use another tool (sftp, scp, ...) to upload your data to the server.



Debugging in RStudio

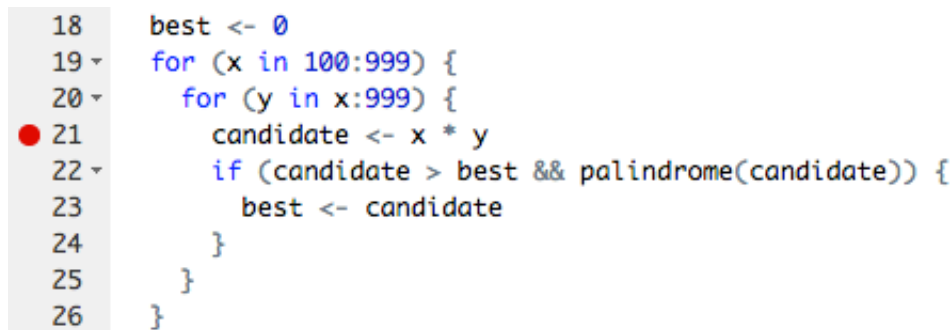
This section is adapted from the more detailed material available on the RStudio website:

<https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio>

Debugging is a broad topic that can be approached in many ways. Generically, at some point you will likely attempt to execute a script in R, receive errors and not know exactly what caused the errors. One approach would be to run your code line by line, but RStudio has some useful [built-in debugging features](#). An in-depth webinar [can be found here](#).

One basic approach to debugging is to create a breakpoint in your code -- this forces your code to "stop"

executing when it reaches some certain function or line number in your code, allowing you then to examine the state of various variables, etc. The easiest way to do this is to set the breakpoint by manually clicking next to the desired line number in the code panel, [per this web example](#):



```
18 best <- 0
19 for (x in 100:999) {
20   for (y in x:999) {
21     candidate <- x * y
22     if (candidate > best && palindrome(candidate)) {
23       best <- candidate
24     }
25   }
26 }
```

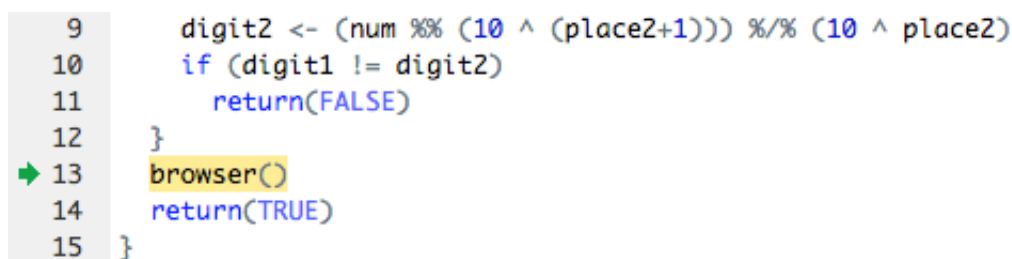
The image shows a snippet of R code in the RStudio editor. A red dot, representing a breakpoint, is placed to the left of line 21, which contains the line `candidate <- x * y`. The code is a nested loop that iterates over values of x and y, calculating a candidate value and checking if it is a palindrome and greater than the current best value.

(image source: <https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio#stopping-on-a-line>)

Setting this **editor breakpoint** creates tracing code associated with the R function object. You can remove the breakpoint by clicking on the red dot by the line number. Also note the Debug toolbar has an option to clear all breakpoints.

Note: keep in mind that you can't set breakpoints anywhere. In general, you want to insert breakpoints at [top-level expressions or simple, named functions](#).

An alternative way to set breakpoints is with the `browser()` function. This must be typed into your code, [per this web example](#):



```
9 digit2 <- (num %% (10 ^ (place2+1))) %% (10 ^ place2)
10 if (digit1 != digit2)
11   return(FALSE)
12 }
13 browser()
14 return(TRUE)
15 }
```

The image shows a snippet of R code. Line 13 contains the `browser()` function call, which is highlighted with a yellow background. A green arrow points to line 13, indicating the execution flow. The code is a function that checks if a number is a palindrome by comparing its digits.

(image source: <https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio#stopping-on-a-line>)

The debugging interface

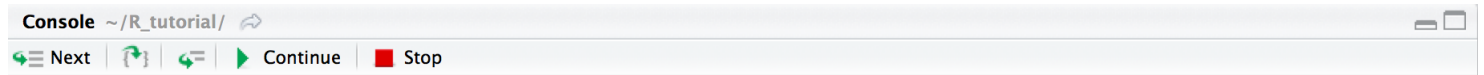
Once your code hits a breakpoint, RStudio enters debugging mode. Details on the debugging interface [can be found here](#), but we summarize the main points below:

The **Environment** tab will display the objects in the environment of the currently executing function (i.e., the function's defined arguments)

The **Traceback** literally traces back how you arrived at the currently executing function (latest executed command is at the top of the list). This is analagous to the "call stack" in other programming languages.

The **Code window** highlights the currently executing function and may create a new tab, named *Source Viewer*, when the current function the debugger is stepping through is not in the main R script.

The **Console** retains most of its normal functionality in debugging mode, but contains some additional buttons that appear at the top to facilitate moving through code lines (see below).



Code Versioning using RStudio

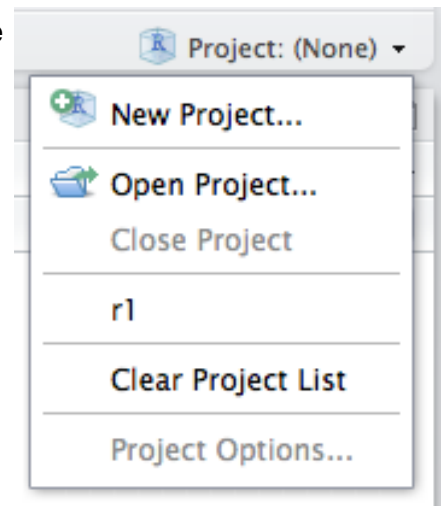
Notes:

- It is assumed that you already know the basics of version control systems. If not, here is a good hands-on introduction to the Git version control system: <https://try.github.io/levels/1/challenges/1>
- It is also assumed that you already have used git on this computer/server. If not, you must first set up your identity; see here for more details: <https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup>.

If you want to use the RStudio interface to control git, you need to create a RStudio Project. To do so, you should first organize your files in one folder that will become your git repository. Once this is done, you can create a new RStudio Project from the upper-right corner of the RStudio IDE window, choosing *New Project*

Now, you have two options:

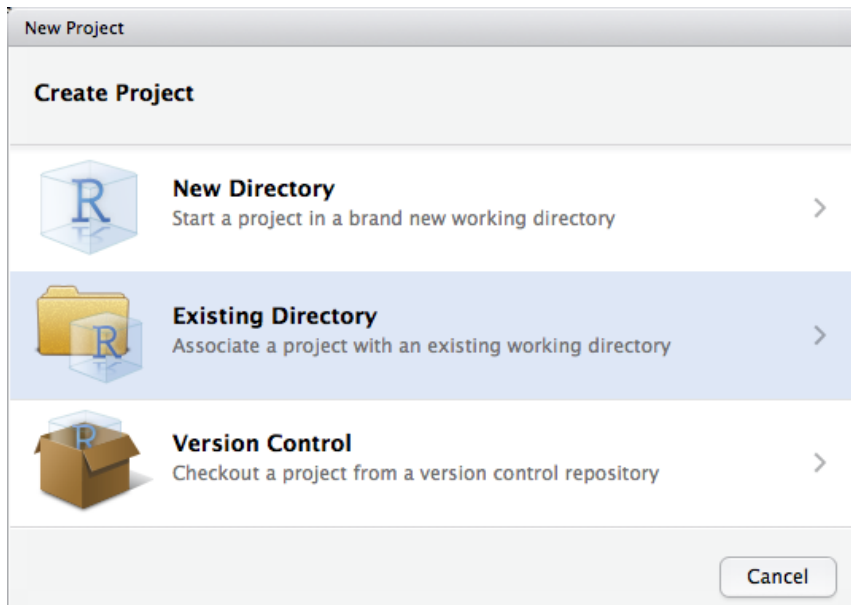
- *Create the Project based on an existing folder* (e.g., you already have been working and writing code and you want to start using git versioning on this work)
- *Create the Project from scratch* based on a new empty folder



Versioning existing work

Create the RStudio Project

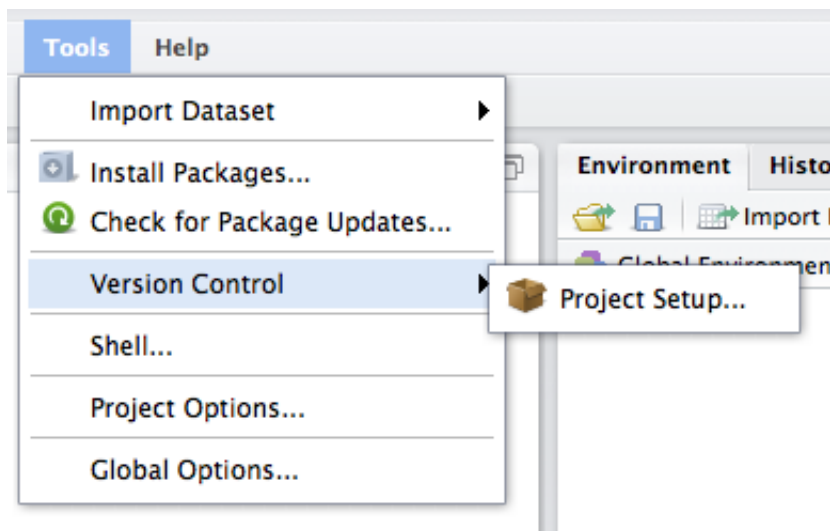
- Choose *Existing Directory* from the wizard



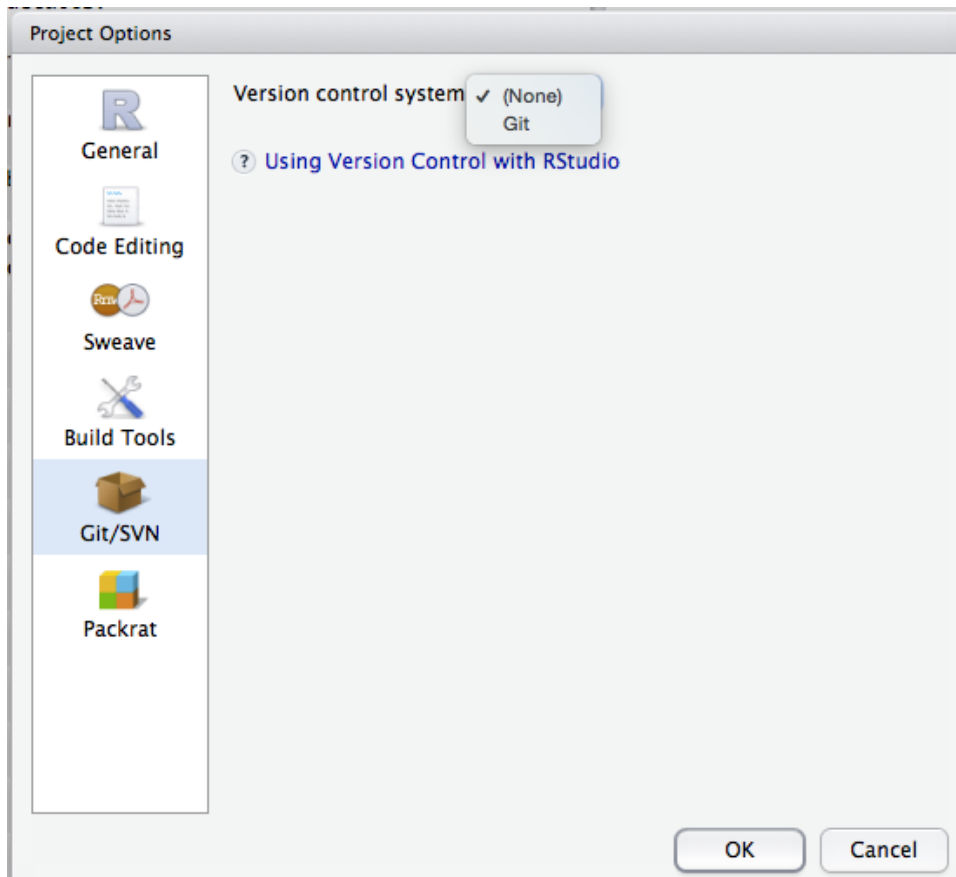
- *Browse* to the main folder containing your data
- *Create Project*

Starting git version control

Now you that have successfully created a RStudio project, you can enable the code versioning. Go to the *Tools* menu -> *Version Control* -> *Project Setup...*



This will open a new window. Change the *None* setting by choosing *Git* from the dropdown menu

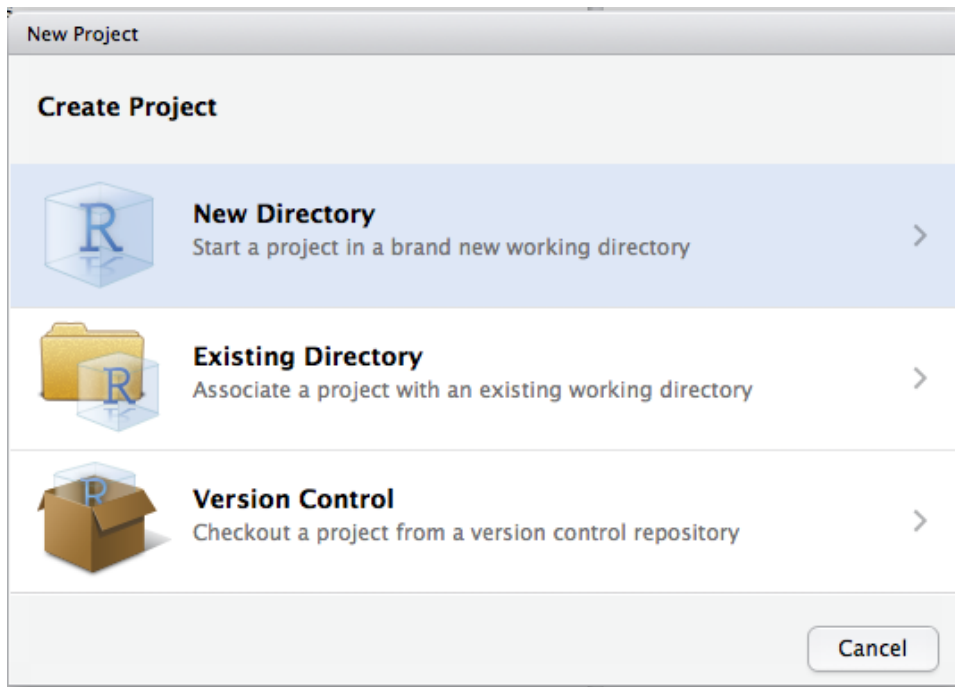


Click *OK*. Answer *Yes* to the 2 following pop-ups to restart RStudio and enable git on your project.

Versioning new work

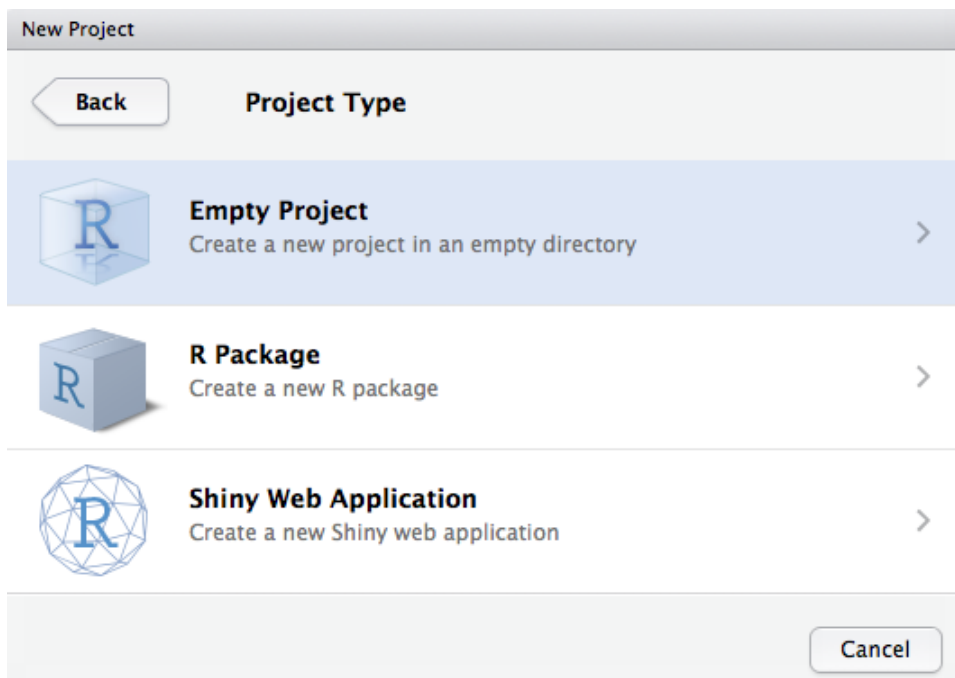
If you are starting to work on a new project for which you have not yet started to write code, you can follow these steps to create a new git repository:

- Create a new RStudio Project from the upper-right corner of the RStudio IDE window, choosing *New Project*
- Choose *New Directory*

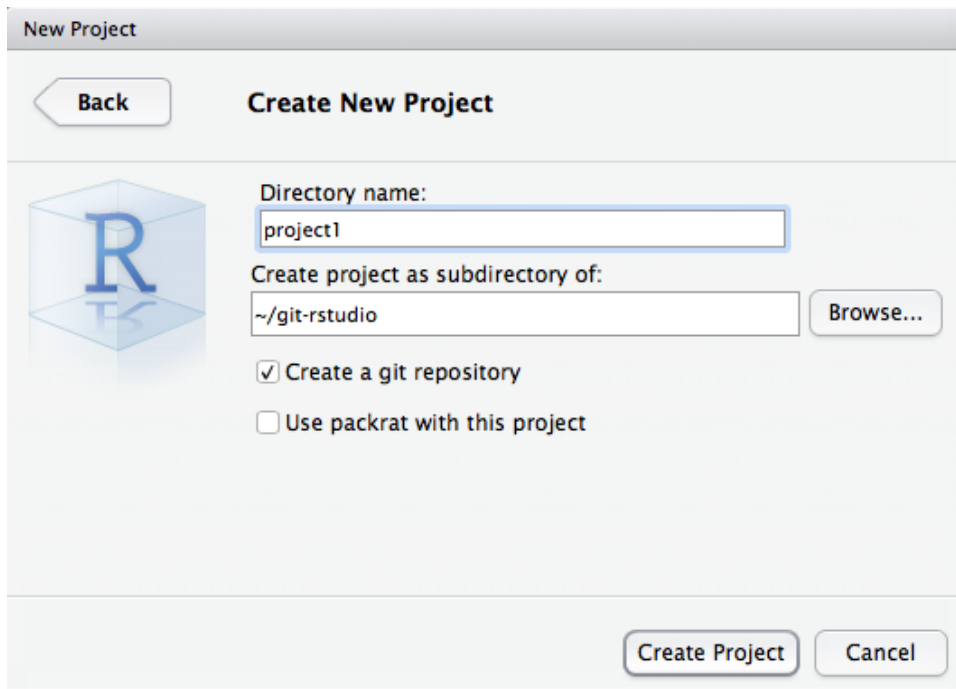


Note: The Version control option at the bottom is only to be chosen when you want to clone an existing repository.

- Choose *Empty Project*



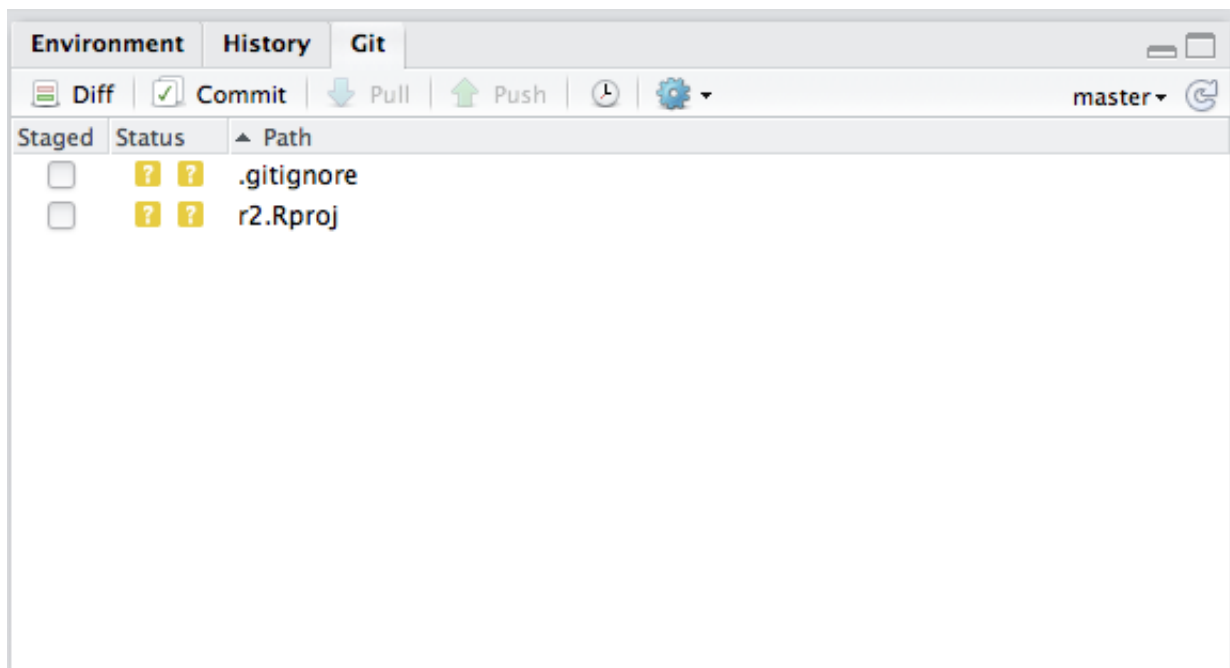
- Give your project (folder) a *name*, *Browse* for its location on your drive. Finally, check the option *Create a git repository* before clicking *Create Project*.



Congratulations, you have a new project with git versioning enabled!!

Using git from RStudio

Now that you have enabled git versioning for your Project, a "Git" tab should have been added to your interface next to "Environment" and "History" tabs

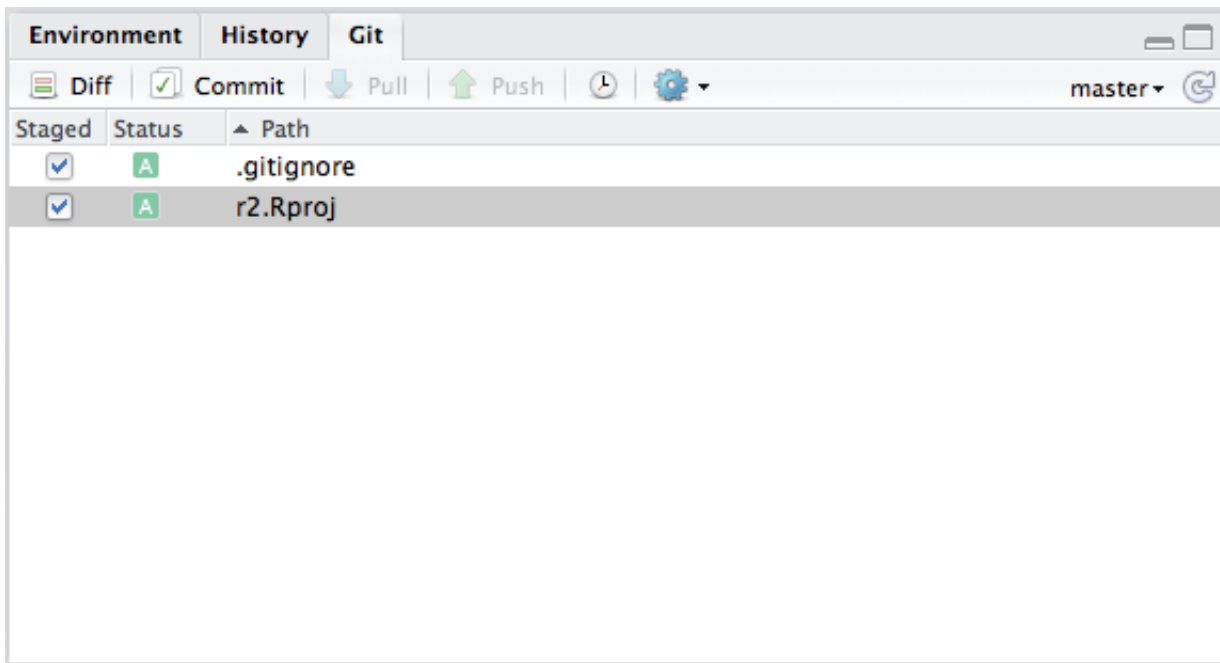


Note that the *.gitignore* file is automatically generated by RStudio. It lists all file formats we do not want to track,

here specifically the temporary files from R and RStudio. You can edit this document to add any type of file you would like git not to track.

Staging

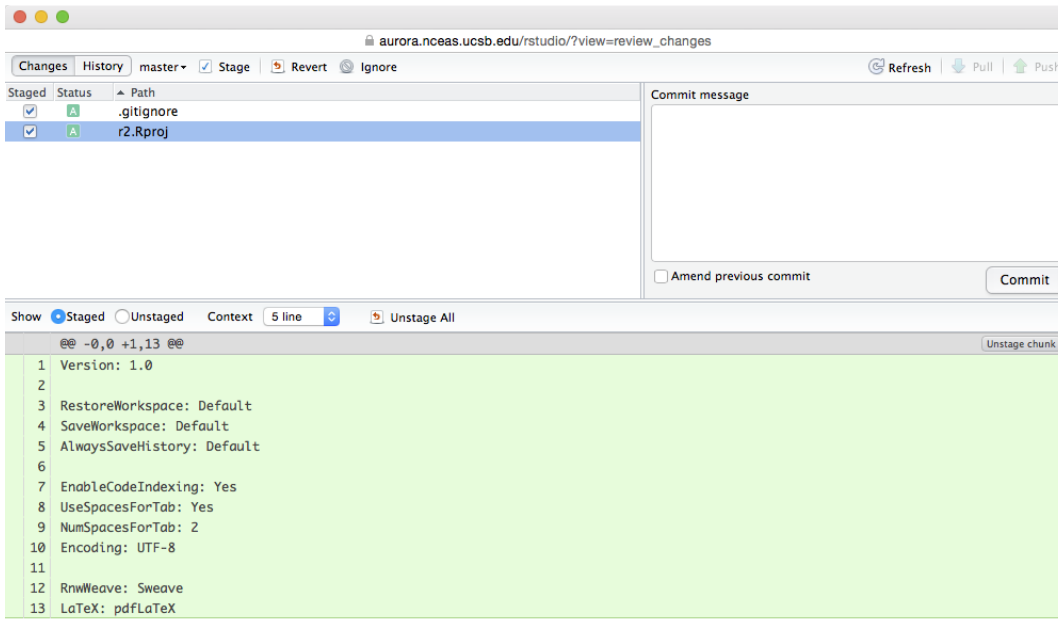
To specify which files you want to track (aka staging), you need to check the boxes in front of the file names. This is the equivalent of the `git add` command.



Committing

Then you can do your first commit -taking a snapshot of the current status of your work for the files you have decided to track (stage) - by clicking on the *Commit* button (above your file names). This is the equivalent of the `git commit` command.

A new window should pop up. Note that certain web browsers will block this pop-up.



You need to write a descriptive message of the work accomplished from the last commit in the Commit message window, then you can click the Commit button (below the Commit message window).

Then you can hit the button *Close* and close the commit window as you would close any system windows.

All the committed files should have disappeared from the *Git* tab on the RStudio IDE. A file will reappear when you save new changes.

And you can start the whole commit process again, once you have reached a milestone in your work!! Commit frequently!! Don't have fear of commitment.

Pulling

Once you have committed your changes, you can `pull` the latest version of the repository from GitHub by clicking on the pull button. Conflict may happen during this steps if several persons have modified the same file. Don't worry, git will walk you through the necessary steps to fix the conflicts (see 3-code-versioning-remote for more info on this). This is the equivalent of the `git pull` command.

Pushing

Once you have finished the pull process, you can click on the `push` button to upload your changes to GitHub and share your work with others. This is the equivalent of the `git push` command.



On the RStudio git panel, the order of the buttons from left to right matches the sequence of actions you need to follow to submit your changes to the GitHub repository!

References

- RStudio IDE cheatsheet: <https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf>
- Using RStudio: <https://support.rstudio.com/hc/en-us/sections/200107586-Using-RStudio>
- RStudio keyboard shortcuts: <https://support.rstudio.com/hc/en-us/articles/200711853-Keybaord-Shortcuts>
- Debugging with R Studio: <https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio>
- Using git from RStudio: <https://support.rstudio.com/hc/en-us/articles/200532077-Version-Control-with-Git-and-SVN>

License

National Center for Ecological Analysis and Synthesis, NCEAS, UCSB

Copyright the Regents of the University of California, 2016



This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).