R and Git(hub)

Michael G. Findley* Michael Denly†

Contents

*Professor, Department of Government, UT Austin, mikefindley@utexas.edu
†PhD Candidate, Department of Government, UT Austin, mdenly@utexas.edu

What is `Git` and Why Use It?

`Git` is a version control system for file management and collaboration. Github is the most common user interface/program for `Git`. It allows people to collaborate or track changes in ways that Dropbox or Microsoft Word cannot.

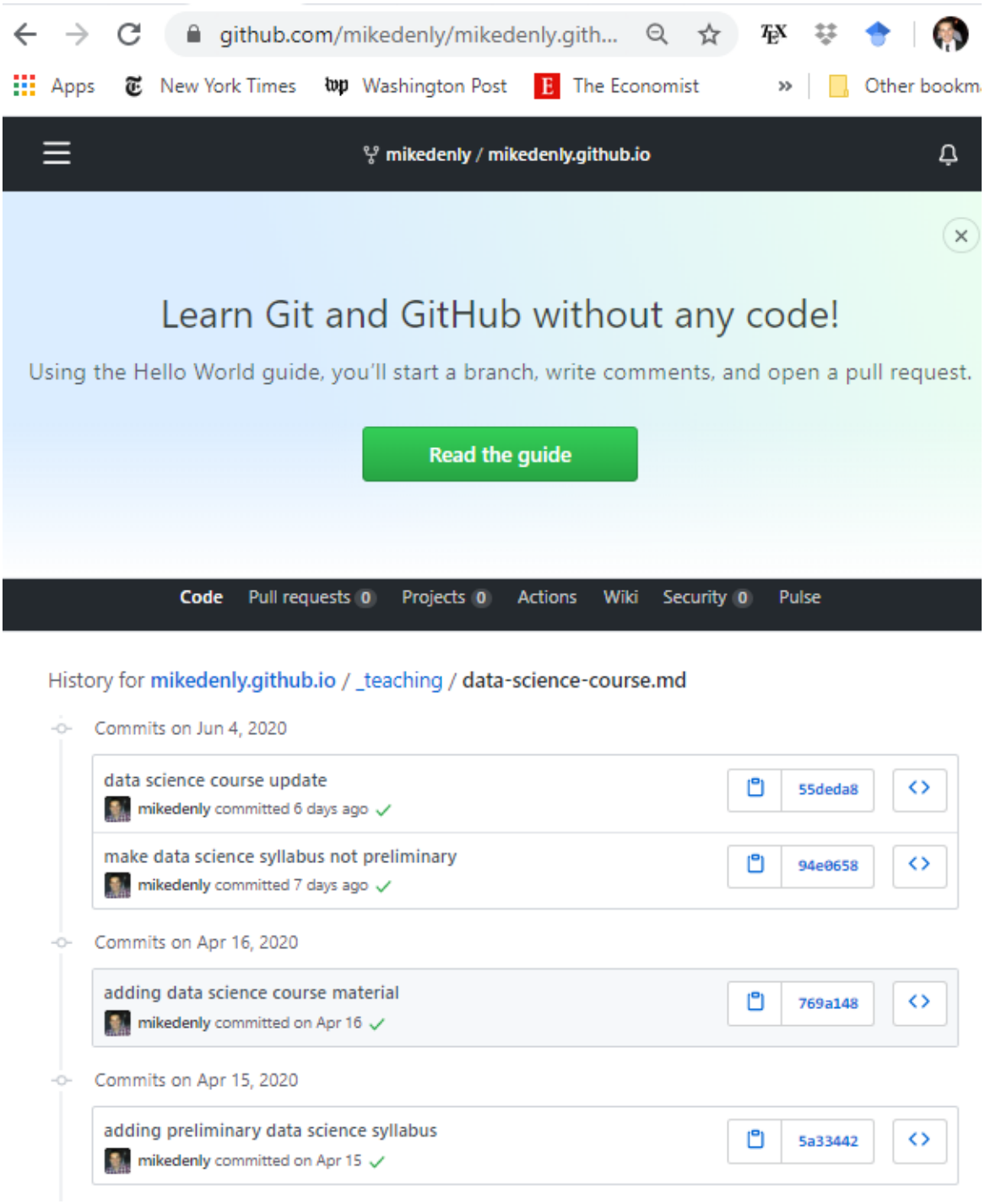
To better understand the power of `Git`, let's use an example. Let's say that you are working on a group project with 5 students for a final paper that requires statistical analysis with `R`.

For your final paper, you have a couple options. First, you could use a Microsoft Word document. The nice part about Microsoft Word is the track changes feature, which most of you know how to use. However, only one person could work on that Word Document at a time. Maybe all team members write different sections and combine them later, but that would be annoying. Therefore, you would probably use a Google Doc–or Overleaf if you needed to work in LaTeX. With Google Docs' new "suggested edits" feature plus the version history, you could work rather seemlessly for writing, accepting changes as you go. Just the same, accepting the individual suggested edits one-by-one is tedious.

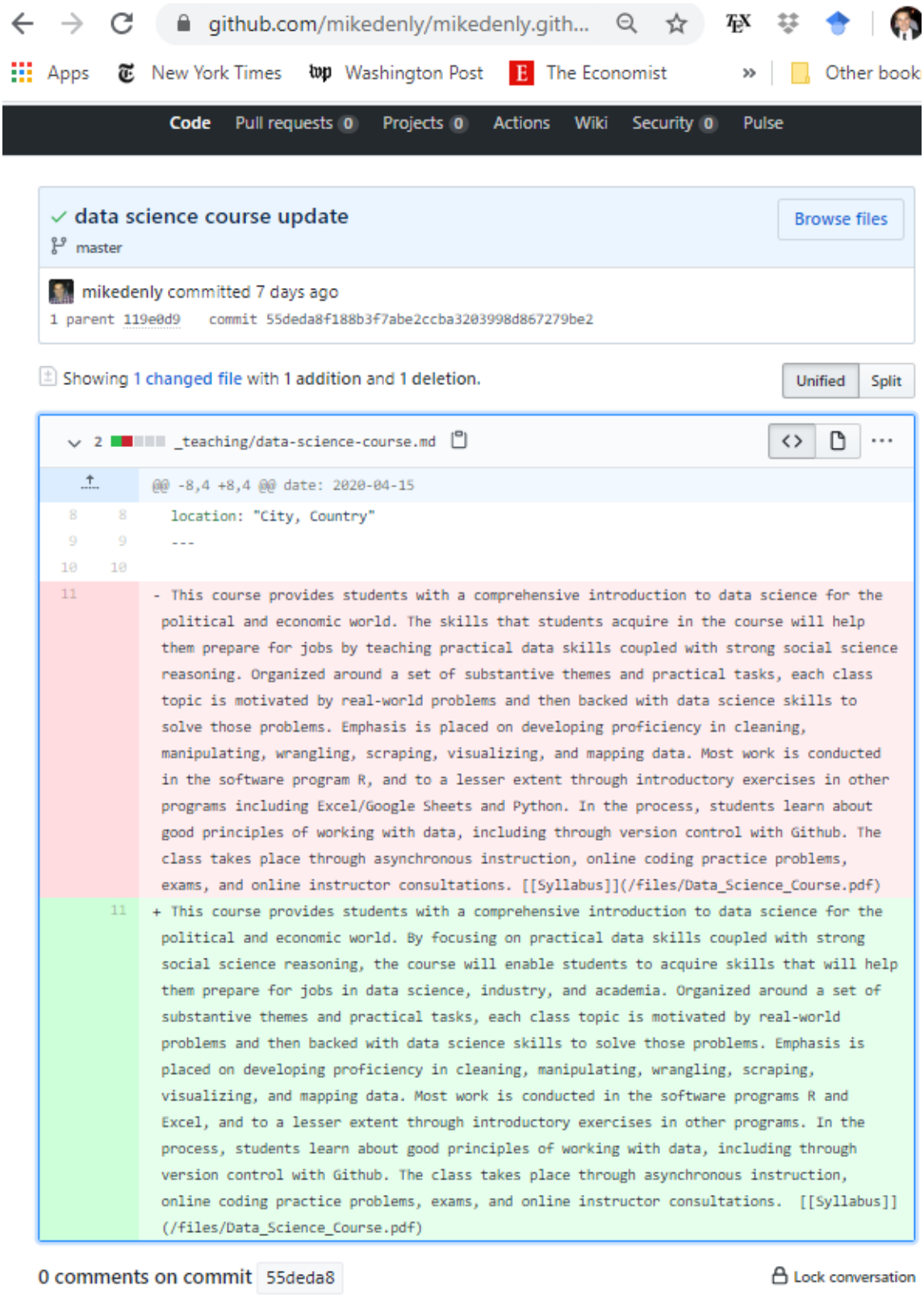
For the `R` part, there is no suggested edits or track changes feature. If Findley, Denly, and Charm were the team members responsible for the `R` part of the final paper, and none of them knew how to use `Git`, each would (a) have to work off separate files and tediously combine them all later; or (b) work off the same file but constantly be in touch over email/text message to ensure that none of them was in the file at the same time to prevent overwriting each others' work.

If Denly, Findley, and Charm had Github accounts and `Git` installed on their computers, then they would have none of these problems. They could easily work off the same file; create branches to separate their work; use a `diff` command to compare each person's changes in each branch against the most updated version of the file; commit the correct answers to the master branch/file; and push their changes to Github. The latter, in turn, could be linked with a folder on their computers, and Github would easily keep track of the different version of the file. In the case where Findley messed everything up, it would be no problem at all. Denly or Charm could easily restore the `R` script back to a previous version. Github keeps a complete record of all the files that were pushed to its server.

`Git` is not only useful for `R`, too. You can use it with Python, Stata, Matlab, LaTeX, etc. Denly also created his website for free using GitHub pages. What did he did was fork (i.e. copy) a template created by someone else directly to his Github account. Then, Denly customized his website using the Markdown language that Professor Findley taught us in the previous submodule. Denly pays nothing for his website–besides 12 dollars per year to Google for the www.mikedenly.com domain name. By contrast, services like Weebly and Wix charge $10-20 per month to host a normal website. Because Denly's website is linked with Github, he can also track the file histories and restore to previous versions, if he wishes. For example, below is the publicly-accessible commit history associated with the teaching page on Denly's website:



Because Denly wrote the page in Markdown, we can also see a tracked history of the changes that he made 7 days ago to the syllabus' introductory paragraph:


github.com/mikedenly/mikedenly.gith...
Apps
New York Times
Washington Post
The Economist
Other book
Code
Pull requests 0
Projects 0
Actions
Wiki
Security 0
Pulse
data science course update
master
Browse files
mikedenly committed 7 days ago
1 parent 119e0d9 commit 55deda8f188b3f7abe2ccba3203998d867279be2
Showing 1 changed file with 1 addition and 1 deletion.
Unified
Split
2 _teaching/data-science-course.md
@@ -8,4 +8,4 @@ date: 2020-04-15
location: "City, Country"

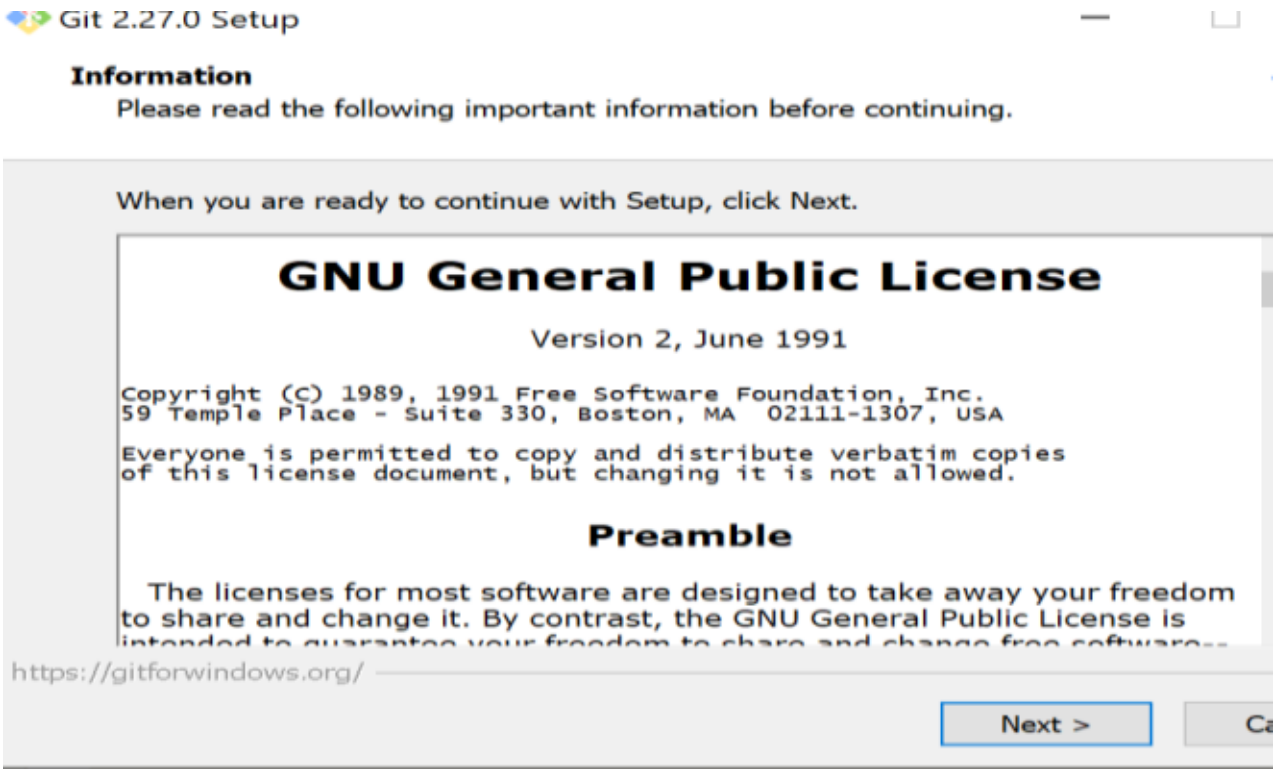
- This course provides students with a comprehensive introduction to data science for the political and economic world. The skills that students acquire in the course will help them prepare for jobs by teaching practical data skills coupled with strong social science reasoning. Organized around a set of substantive themes and practical tasks, each class topic is motivated by real-world problems and then backed with data science skills to solve those problems. Emphasis is placed on developing proficiency in cleaning, manipulating, wrangling, scraping, visualizing, and mapping data. Most work is conducted in the software program R, and to a lesser extent through introductory exercises in other programs including Excel/Google Sheets and Python. In the process, students learn about good principles of working with data, including through version control with Github. The class takes place through asynchronous instruction, online coding practice problems, exams, and online instructor consultations. [[Syllabus]](/files/Data_Science_Course.pdf)
+ This course provides students with a comprehensive introduction to data science for the political and economic world. By focusing on practical data skills coupled with strong social science reasoning, the course will enable students to acquire skills that will help them prepare for jobs in data science, industry, and academia. Organized around a set of substantive themes and practical tasks, each class topic is motivated by real-world problems and then backed with data science skills to solve those problems. Emphasis is placed on developing proficiency in cleaning, manipulating, wrangling, scraping, visualizing, and mapping data. Most work is conducted in the software programs R and Excel, and to a lesser extent through introductory exercises in other programs. In the process, students learn about good principles of working with data, including through version control with Github. The class takes place through asynchronous instruction, online coding practice problems, exams, and online instructor consultations. [[Syllabus]](/files/Data_Science_Course.pdf)
0 comments on commit 55deda8
Lock conversation

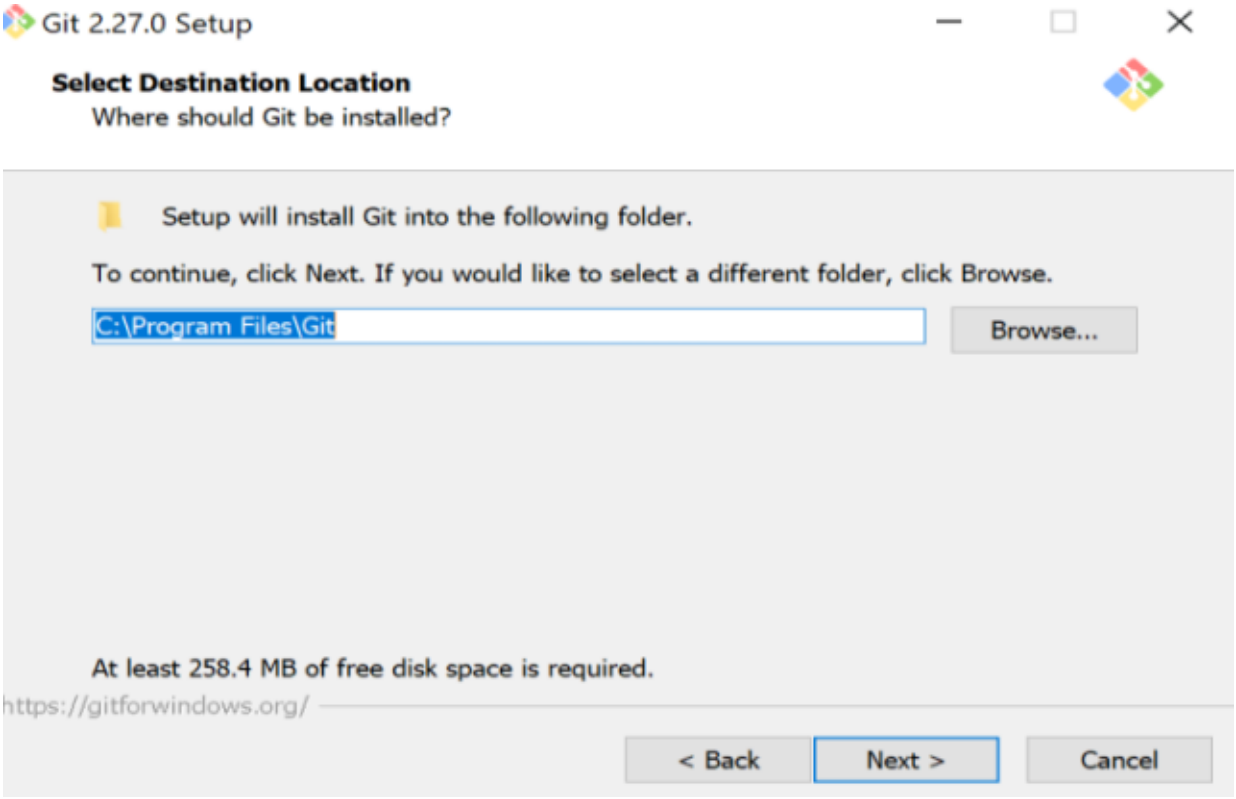
Getting Started with `Git`, (Potentially) `GitBash`, and Github

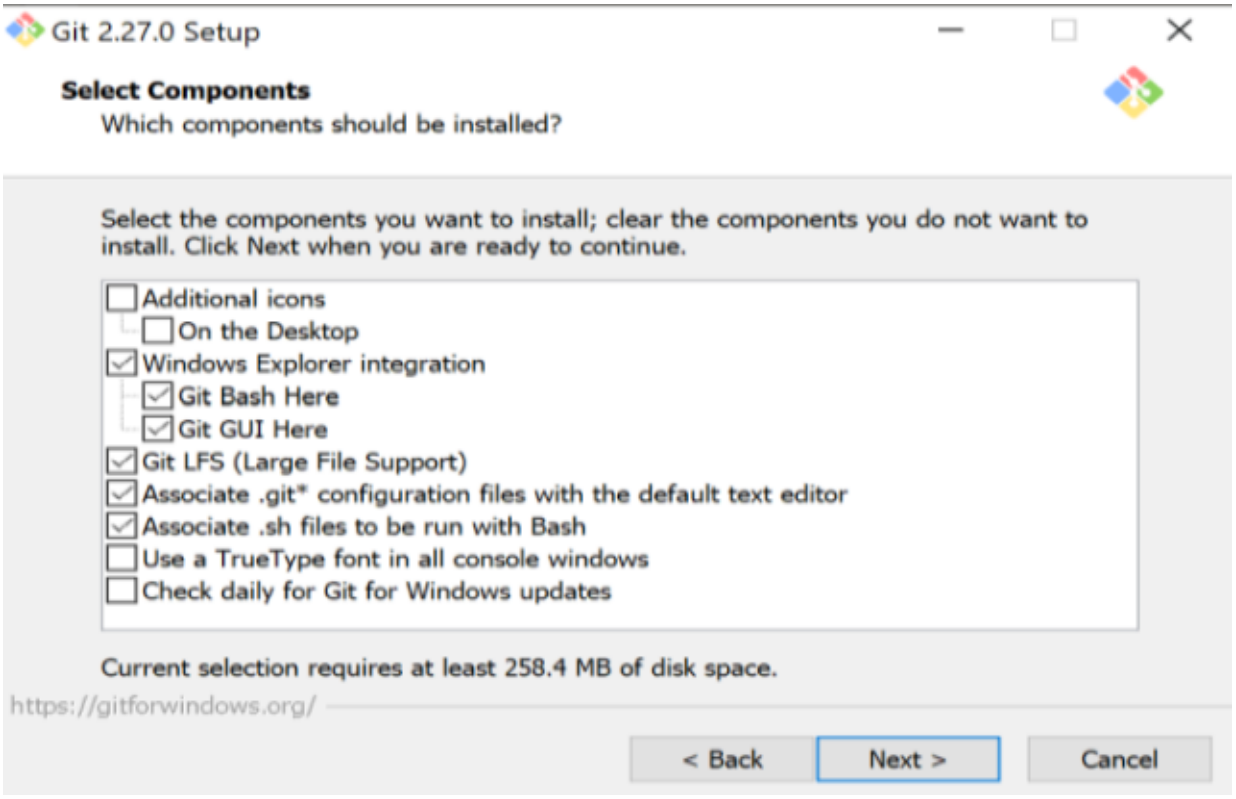
Setting up `Git` and `GitBash` on Windows

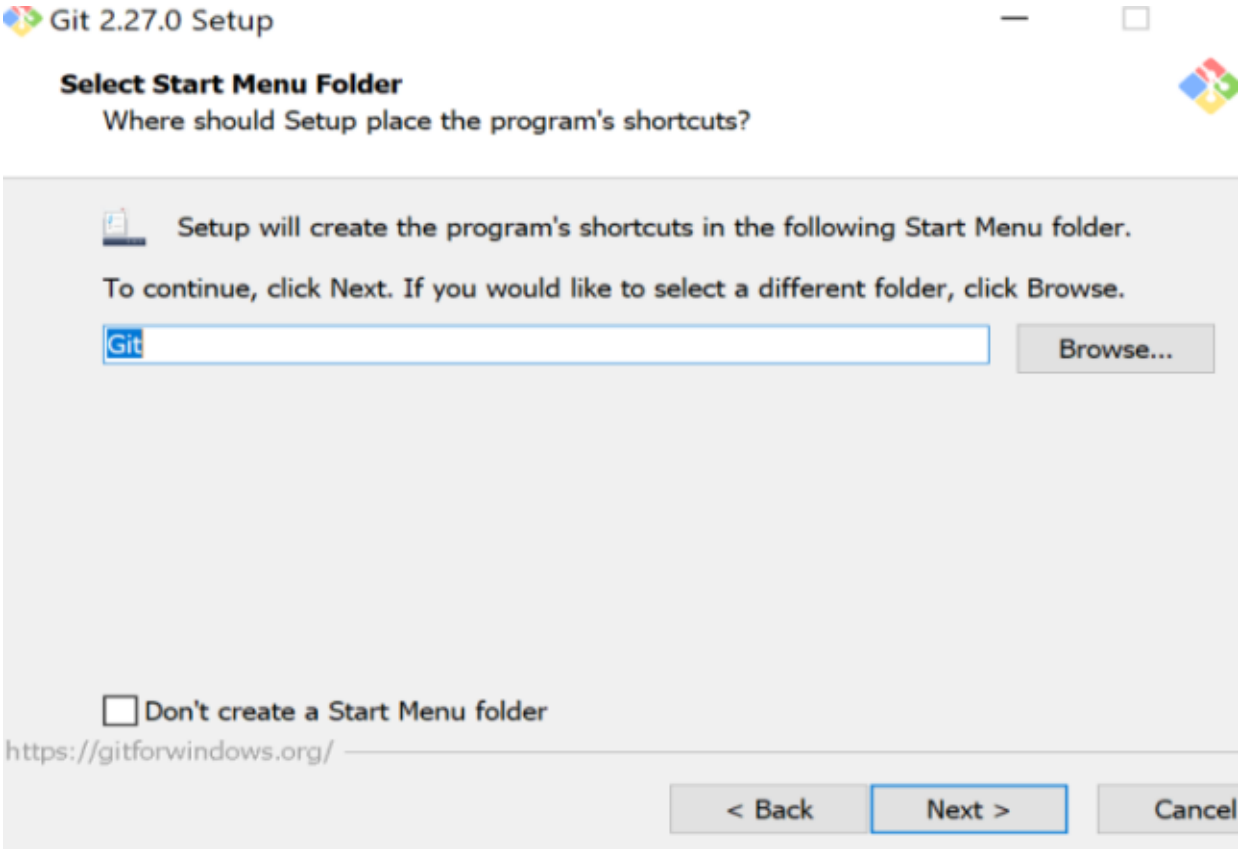
Nowadays, it possible to install `Git` and `GitBash` together. To do so, go to https://git-scm.com/downloads and install `Git` for your respective operating system. If you are wondering about the 32-bit or 64-bit distinction, it depends on the operating system that your computer is using. The 32-bit system is a bit slower, but it will still work on a 64-bit computer. Where to find what operating system you have depends on your computer, so we would suggest Googling accordingly.

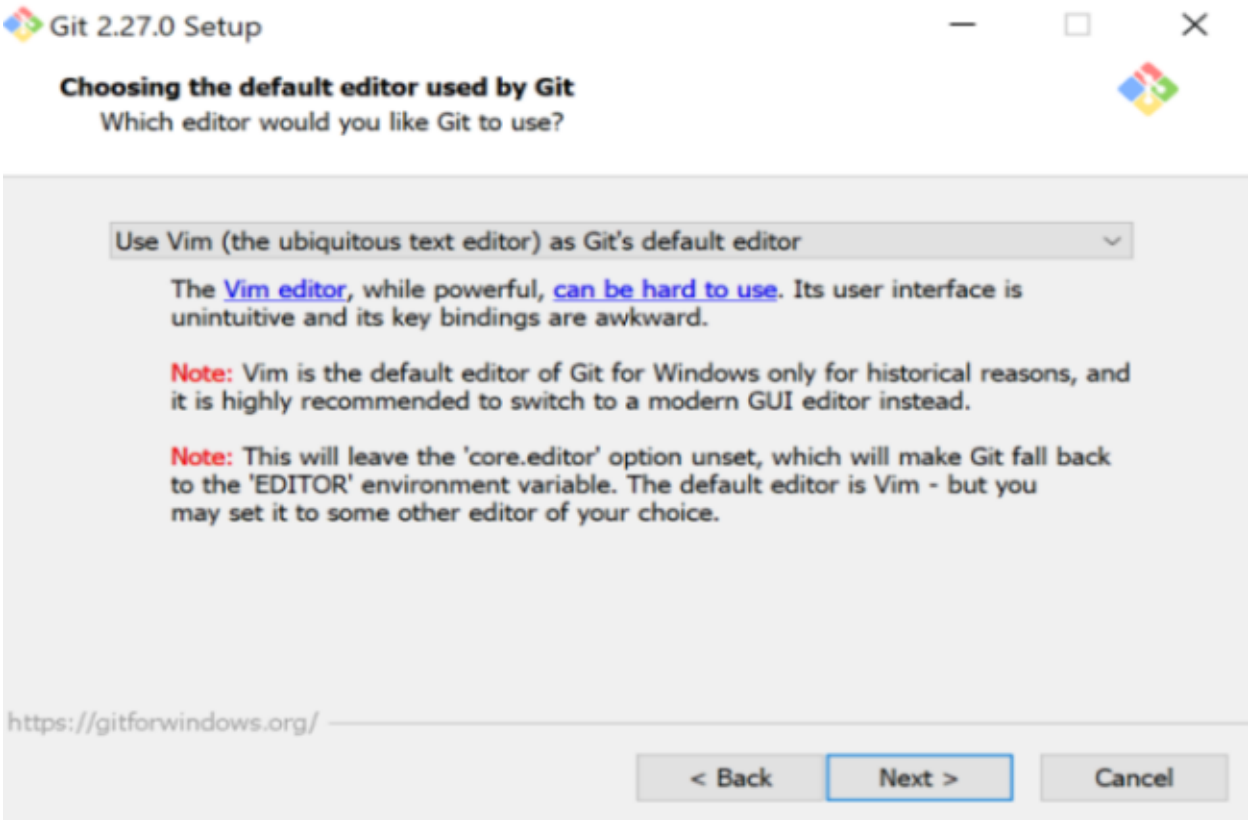
The actual installation is a bit complicated, so we will provide screen shots below.

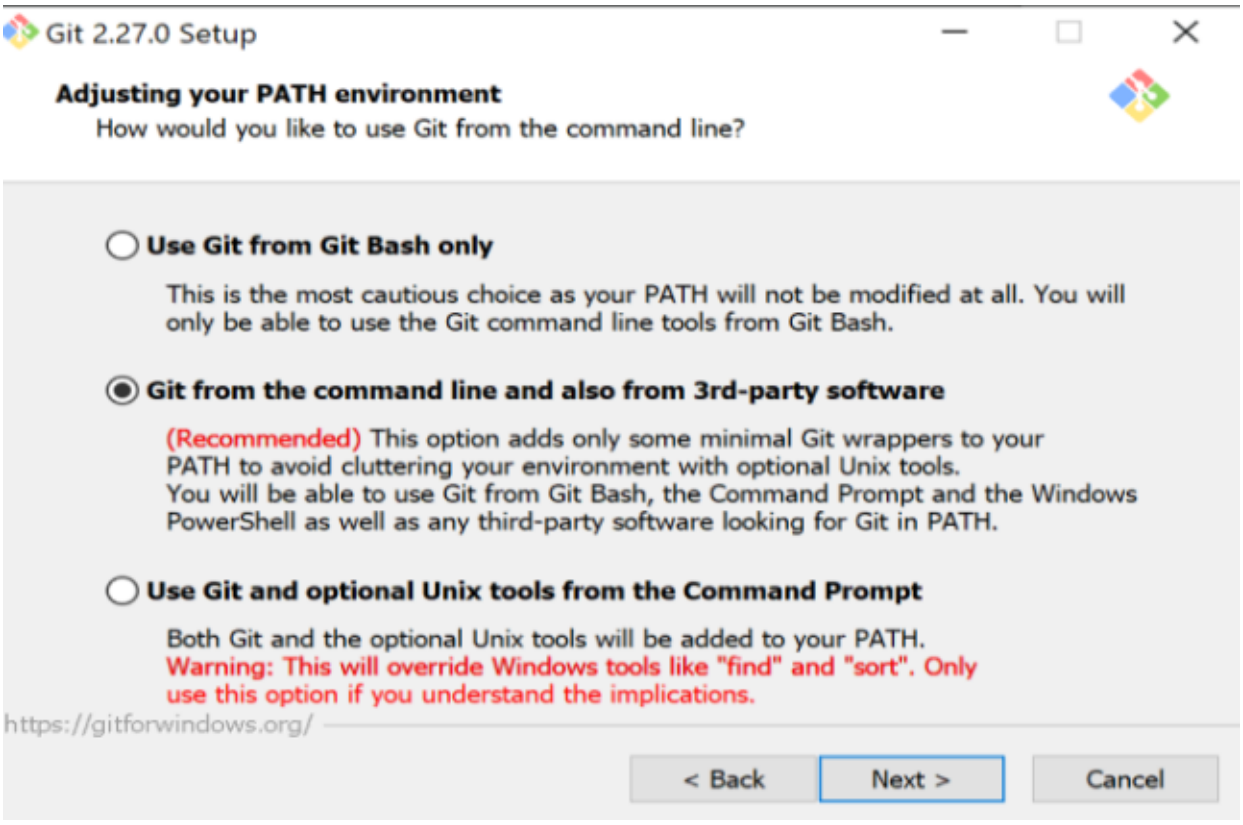


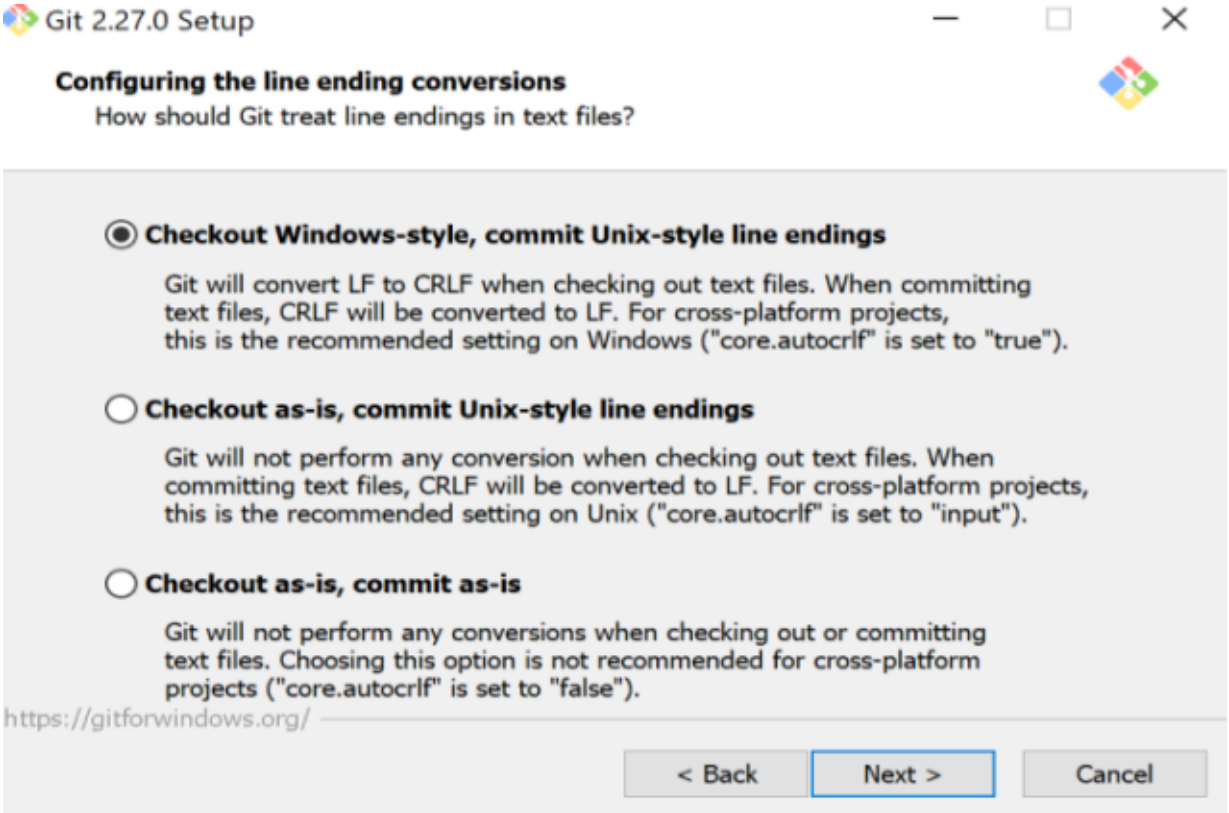

Git 2.27.0 Setup
Select Destination Location
Where should Git be installed?
Setup will install Git into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
C:\Program Files\Git
Browse...
At least 258.4 MB of free disk space is required.
https://gitforwindows.org/
< Back
Next >
Cancel

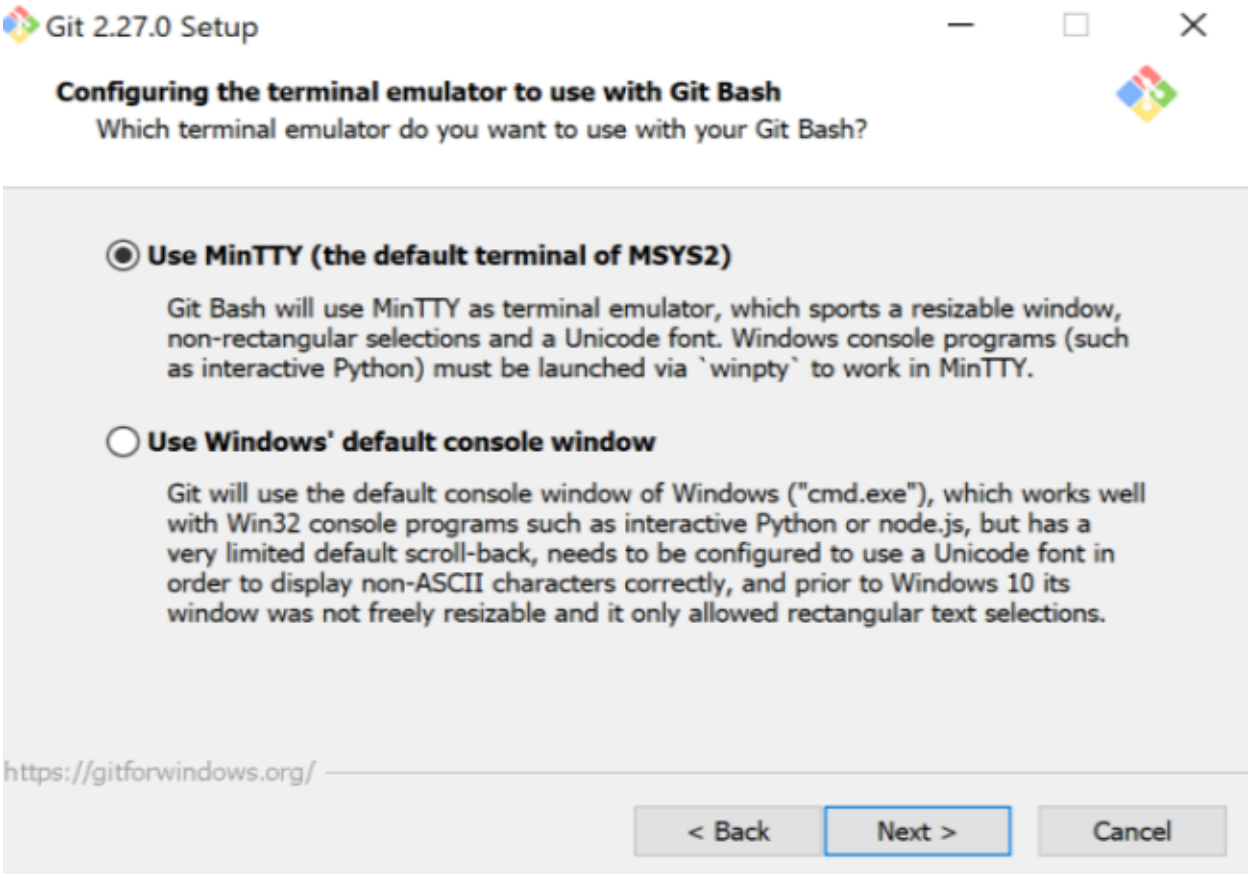

Git 2.27.0 Setup
Select Components
Which components should be installed?
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
Additional icons
On the Desktop
Windows Explorer integration
Git Bash Here
Git GUI Here
Git LFS (Large File Support)
Associate .git* configuration files with the default text editor
Associate .sh files to be run with Bash
Use a TrueType font in all console windows
Check daily for Git for Windows updates
Current selection requires at least 258.4 MB of disk space.
https://gitforwindows.org/
< Back
Next >
Cancel

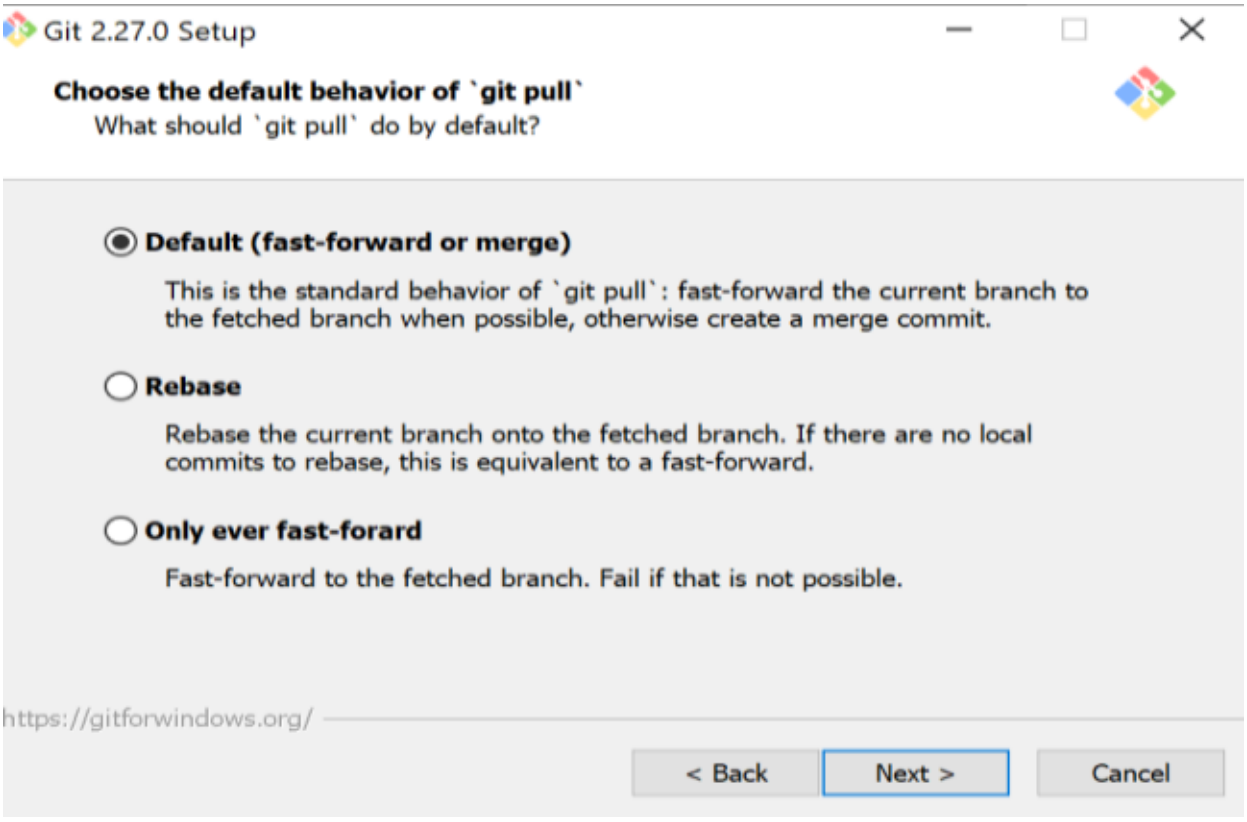

Git 2.27.0 Setup
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
Git
Browse...
Don't create a Start Menu folder
https://gitforwindows.org/
< Back
Next >
Cancel

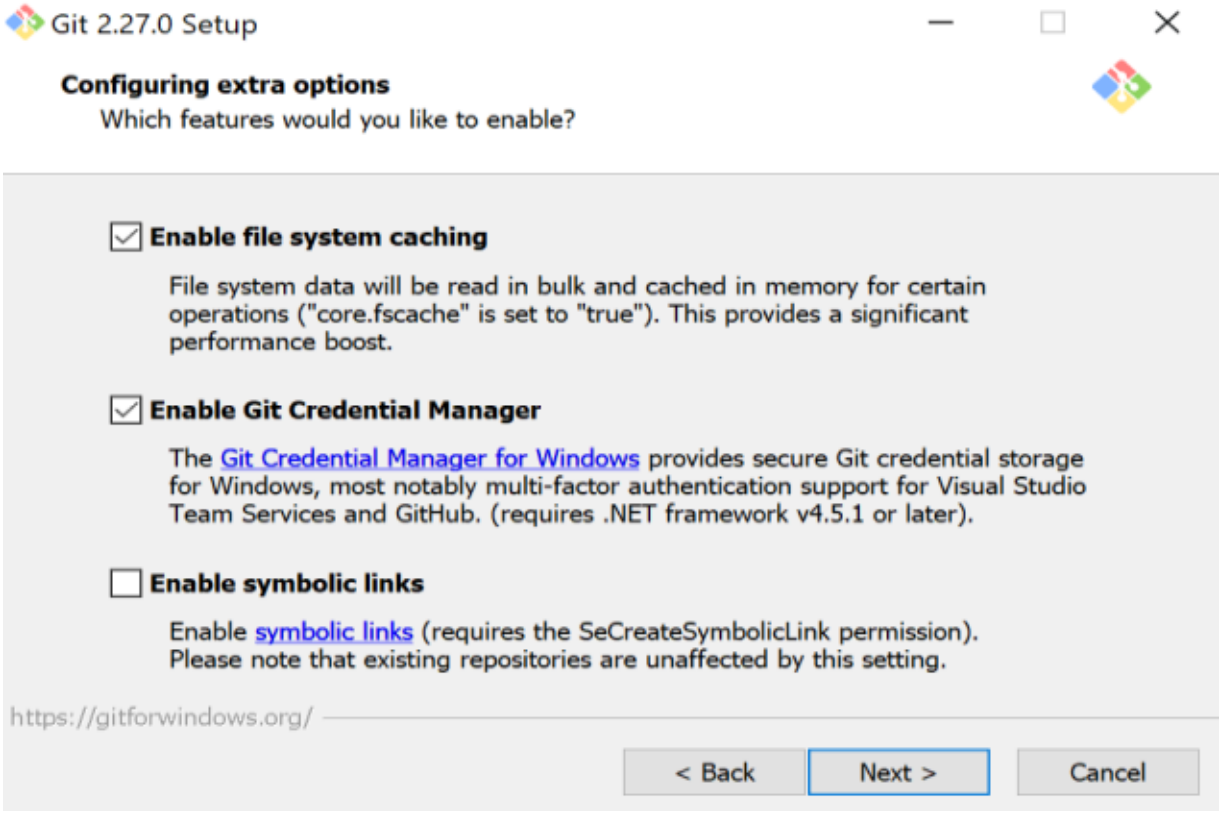

Git 2.27.0 Setup
Choosing the default editor used by Git
Which editor would you like Git to use?
Use Vim (the ubiquitous text editor) as Git's default editor
The Vim editor, while powerful, can be hard to use. Its user interface is unintuitive and its key bindings are awkward.
Note: Vim is the default editor of Git for Windows only for historical reasons, and it is highly recommended to switch to a modern GUI editor instead.
Note: This will leave the 'core.editor' option unset, which will make Git fall back to the 'EDITOR' environment variable. The default editor is Vim - but you may set it to some other editor of your choice.
https://gitforwindows.org/
< Back
Next >
Cancel

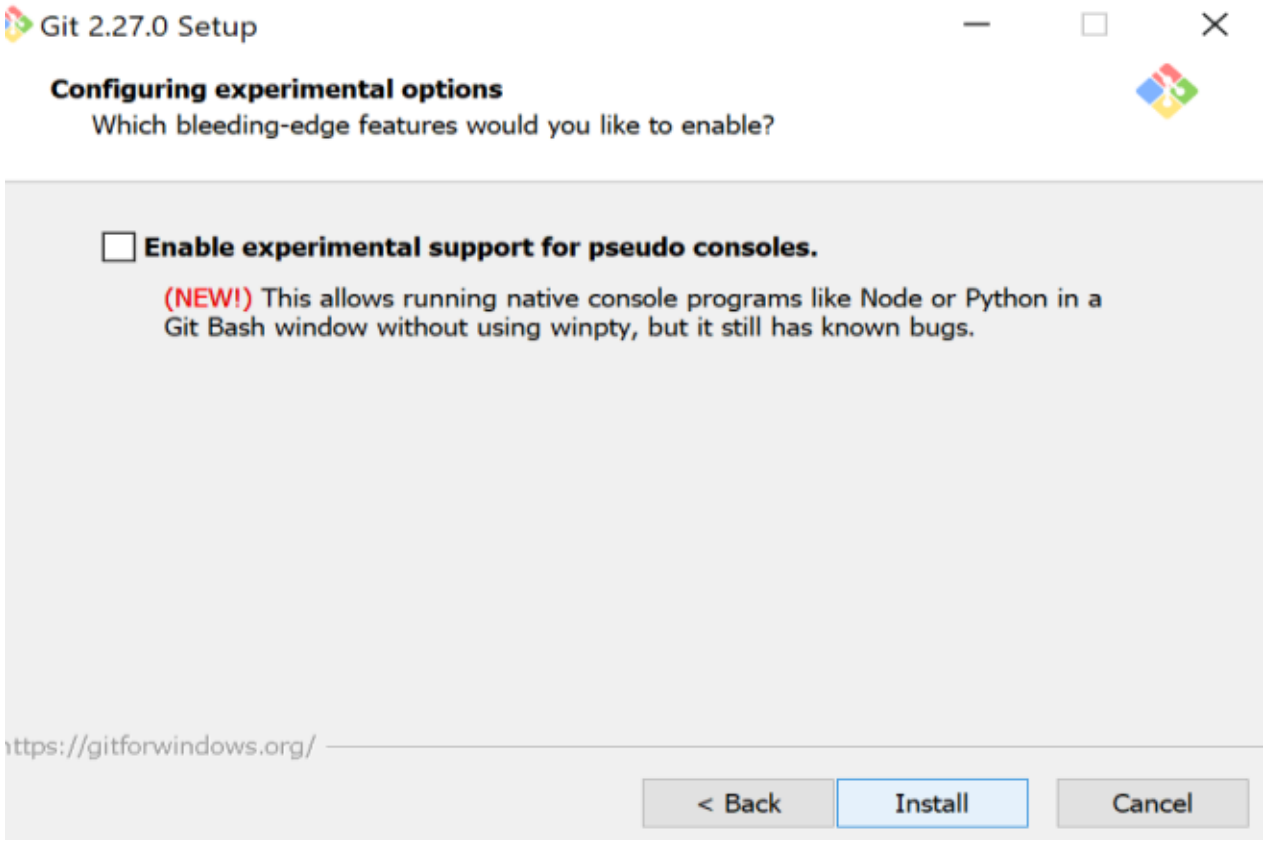

Git 2.27.0 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
Git from the command line and also from 3rd-party software
(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.
Use Git and optional Unix tools from the Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.27.0 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.27.0 Setup
Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?
Use MinTTY (the default terminal of MSYS2)
Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.
Use Windows' default console window
Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.27.0 Setup
Choose the default behavior of `git pull`
What should `git pull` do by default?
Default (fast-forward or merge)
This is the standard behavior of `git pull`: fast-forward the current branch to the fetched branch when possible, otherwise create a merge commit.
Rebase
Rebase the current branch onto the fetched branch. If there are no local commits to rebase, this is equivalent to a fast-forward.
Only ever fast-forard
Fast-forward to the fetched branch. Fail if that is not possible.
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.27.0 Setup
Configuring extra options
Which features would you like to enable?
Enable file system caching
File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.
Enable Git Credential Manager
The Git Credential Manager for Windows provides secure Git credential storage for Windows, most notably multi-factor authentication support for Visual Studio Team Services and GitHub. (requires .NET framework v4.5.1 or later).
Enable symbolic links
Enable symbolic links (requires the SeCreateSymbolicLink permission).
Please note that existing repositories are unaffected by this setting.
https://gitforwindows.org/
< Back
Next >
Cancel

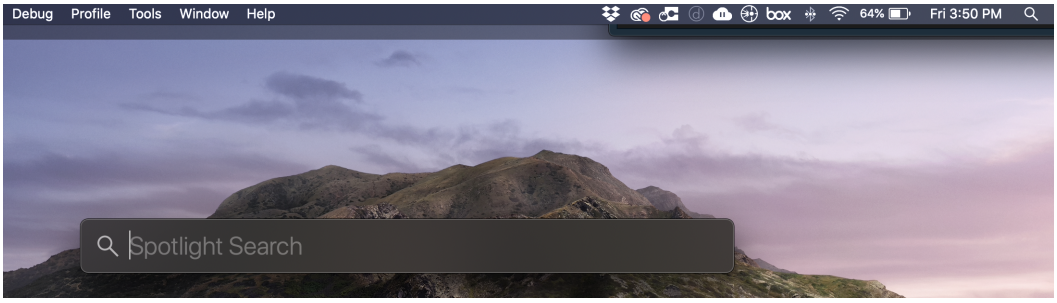

Git 2.27.0 Setup
Configuring experimental options
Which bleeding-edge features would you like to enable?
Enable experimental support for pseudo consoles.
(NEW!) This allows running native console programs like Node or Python in a Git Bash window without using winpty, but it still has known bugs.
https://gitforwindows.org/
< Back
Install
Cancel

Setting up `Git` on a Mac

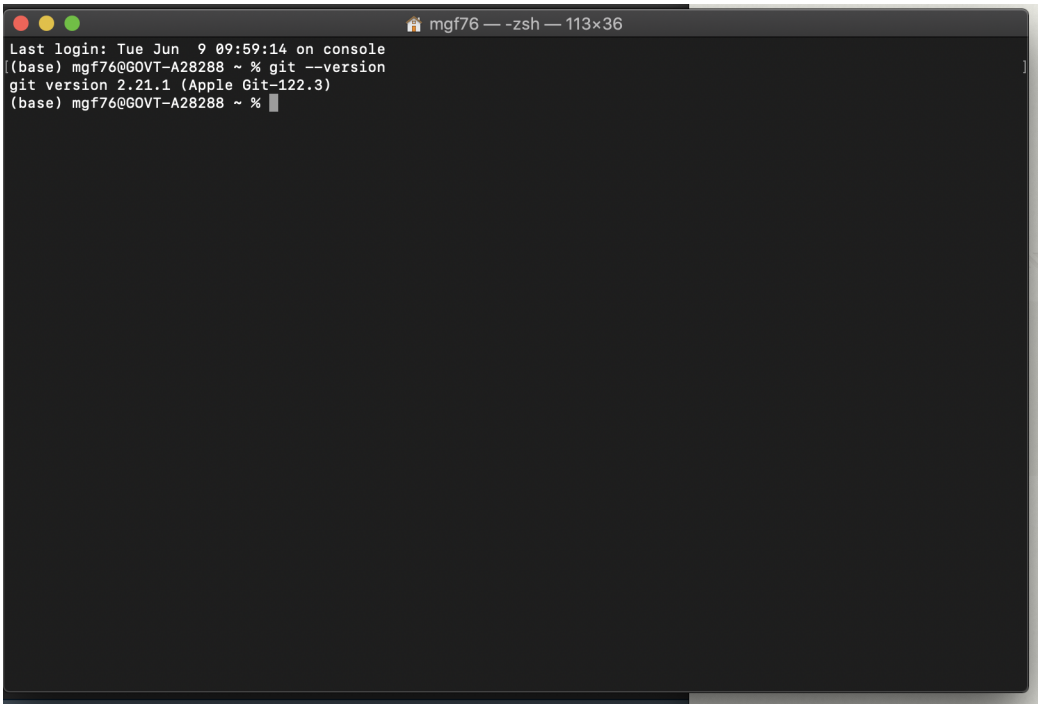
`Git` comes preinstalled on Mac, but you should check to see whether you have the latest version. First open the terminal:

- Click on magnifying glass in top right and then type "terminal"
- Or do `Cmd+Space bar` and type "terminal"

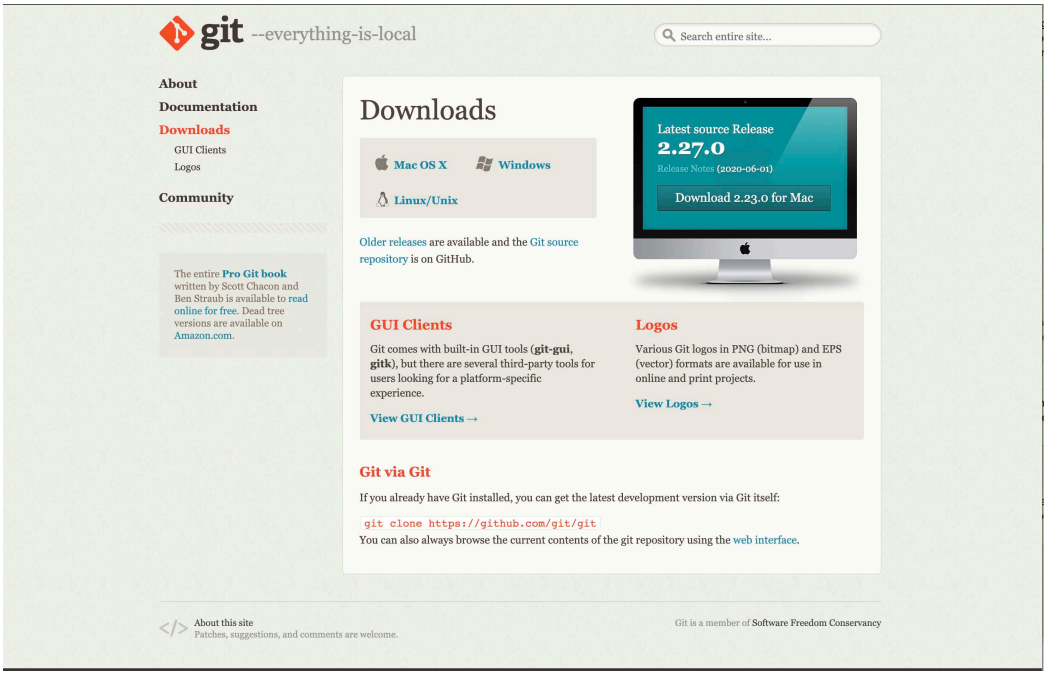
See screenshot here for where you'd do this. You should be pretty familiar already with the spotlight function:



In terminal, type: `git --version`. See screenshot:

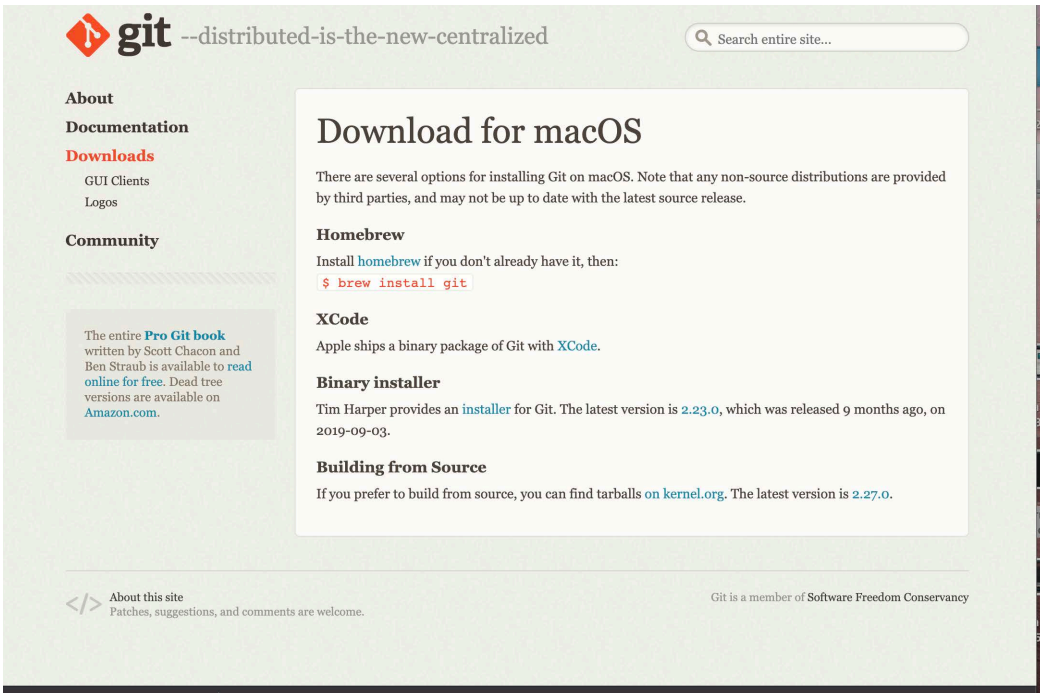


If nothing appears, then you don't have it. This could occur on an old Mac. If you don't have it, then go to the git download page linked [here]

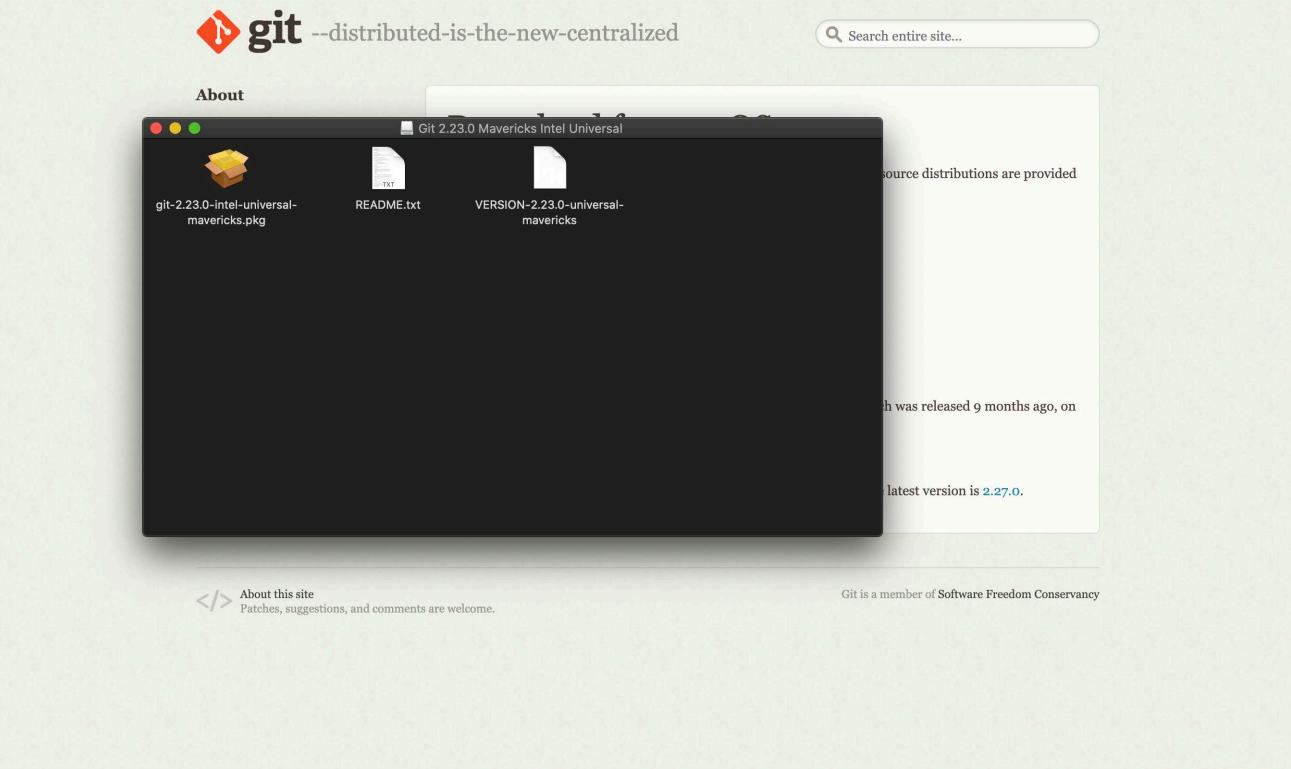


A few screenshots show you where to go from here.

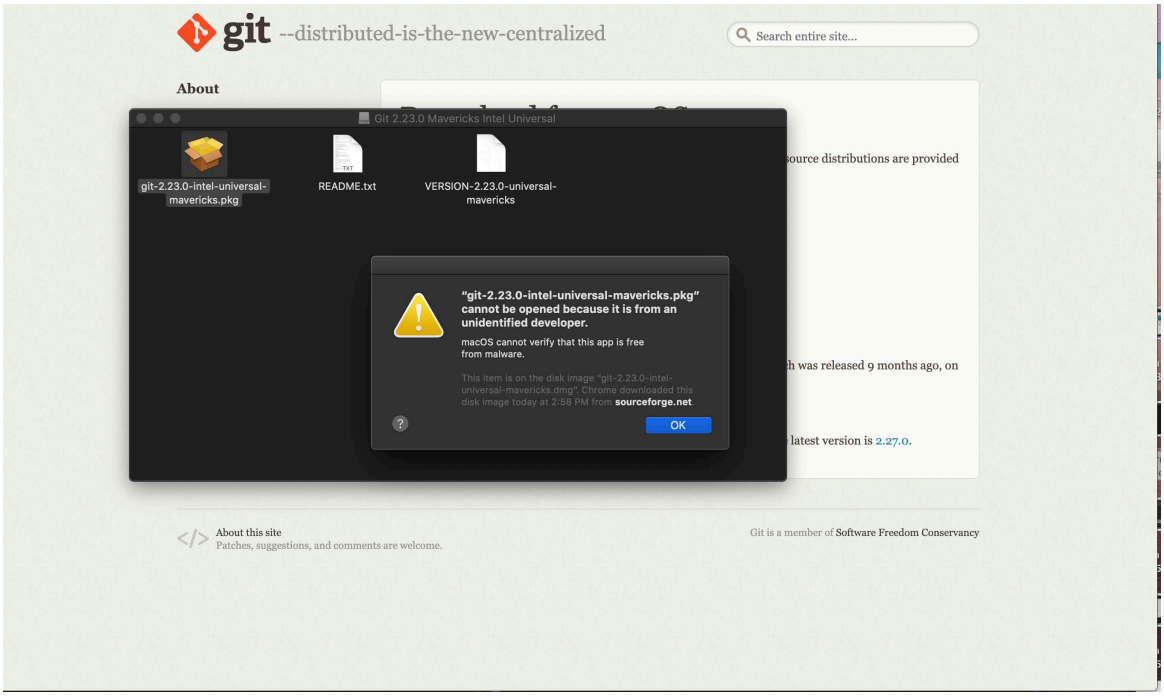
First, when you click the download page, you'll get four options. The easiest is to choose 2.23.0 under the Binary Installer option.



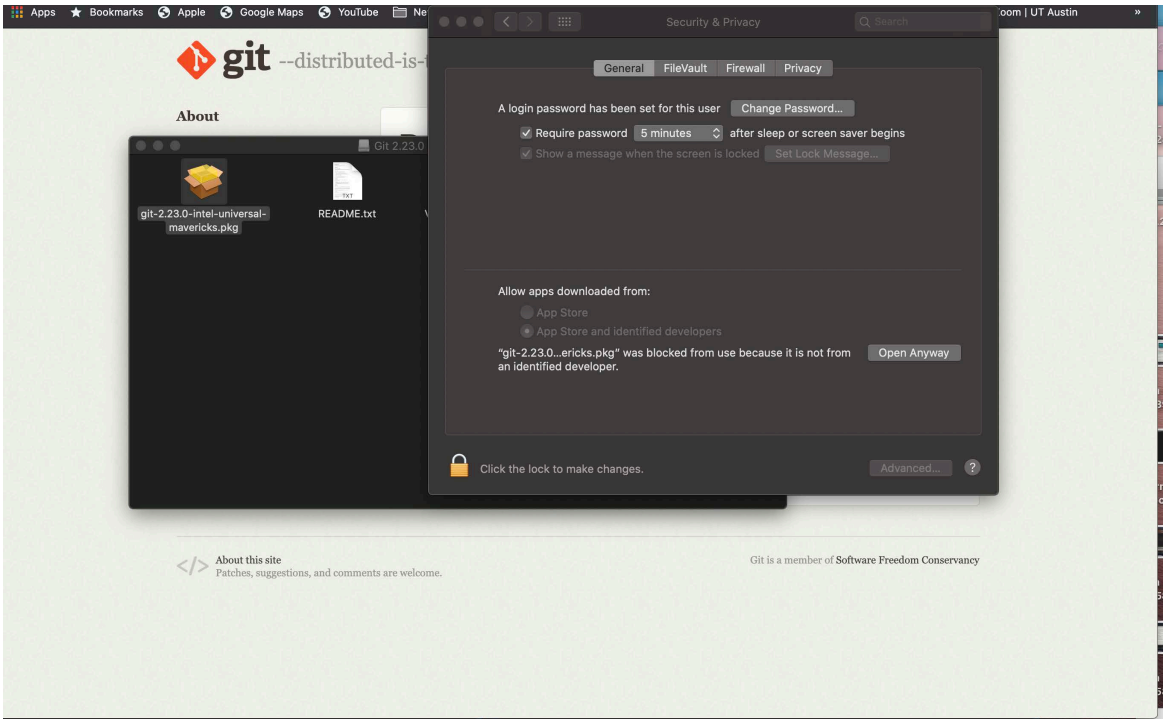
You should download that and then the following will appear. Double click on the .pkg icon and then follow the steps.



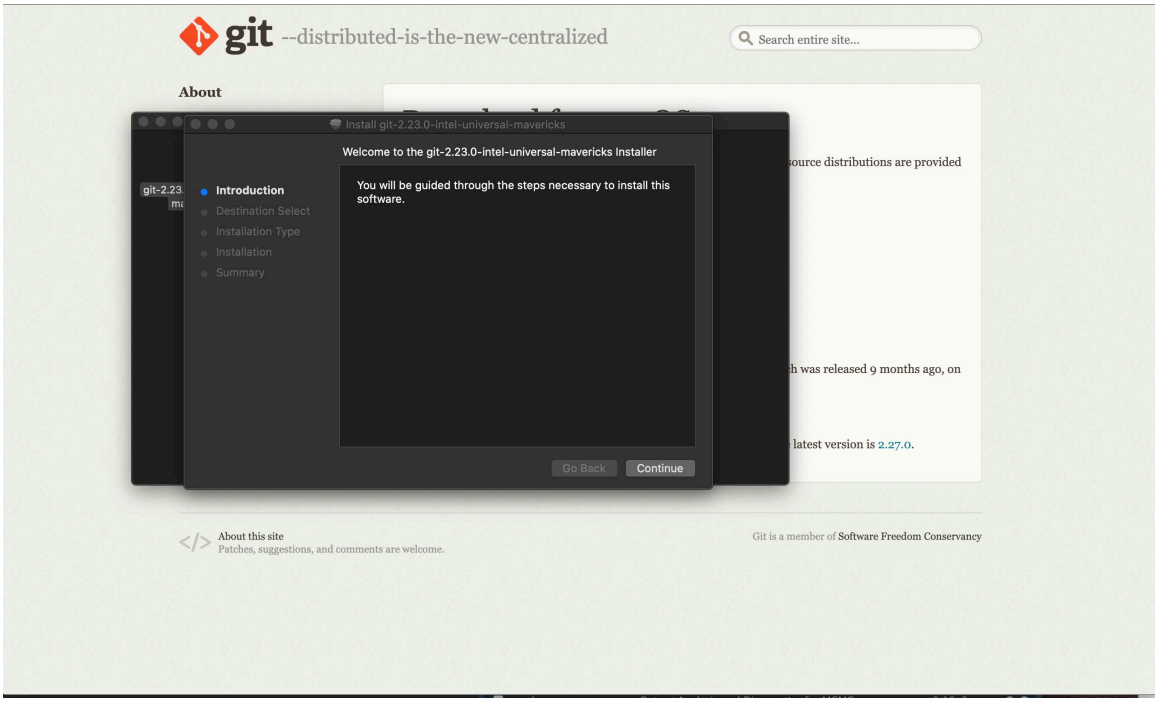
Note that your computer might be configured to deny access to files like this.



So you may need to go into Settings. The **Gear** icon in your toolbar. Then click on **Security and Privacy**. Then click on **General**. And then enable from the prompt at the bottom.



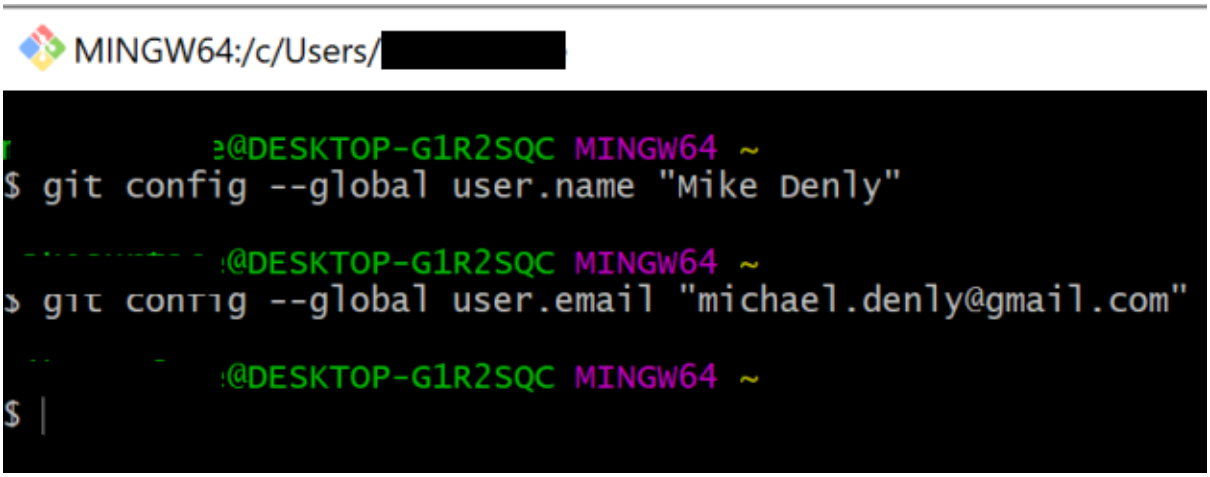
Once you've done all of that, you should be able to finish the installation.



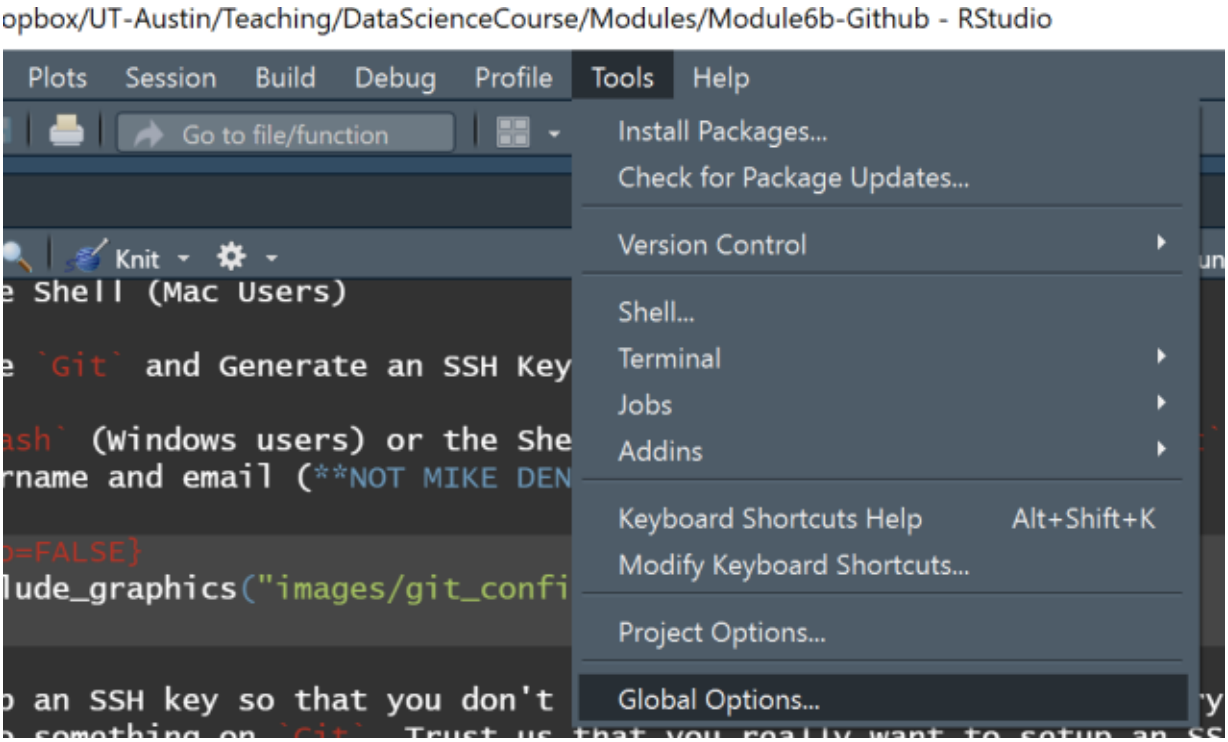
If you get a version #, just double check to be sure it's the latest. If you want to, go ahead and download the latest version. You can use an older version but you don't want to get too far out of date. You might need to restart your computer, then go to terminal (same steps as above) and check the version.

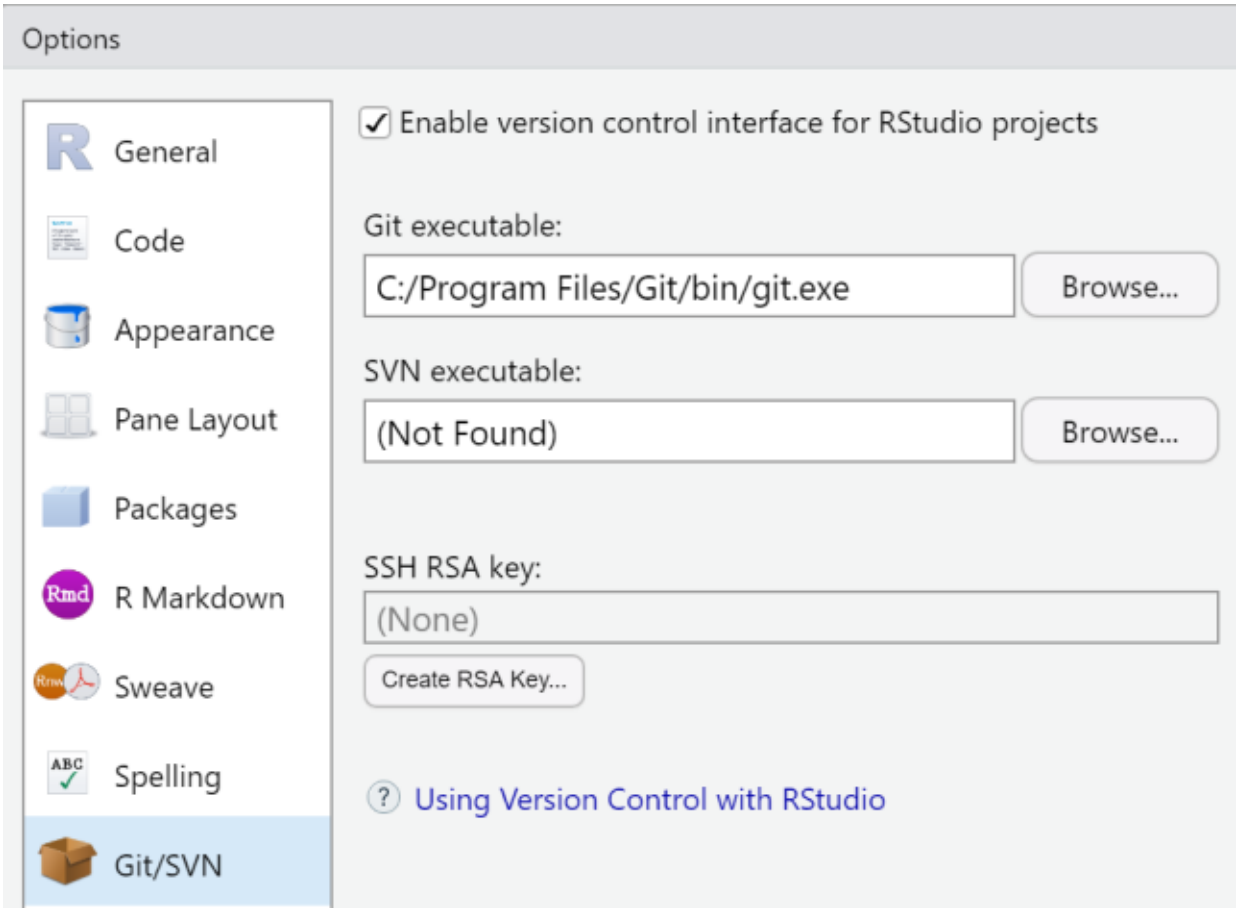
Configure `Git` and Generate an SSH Key

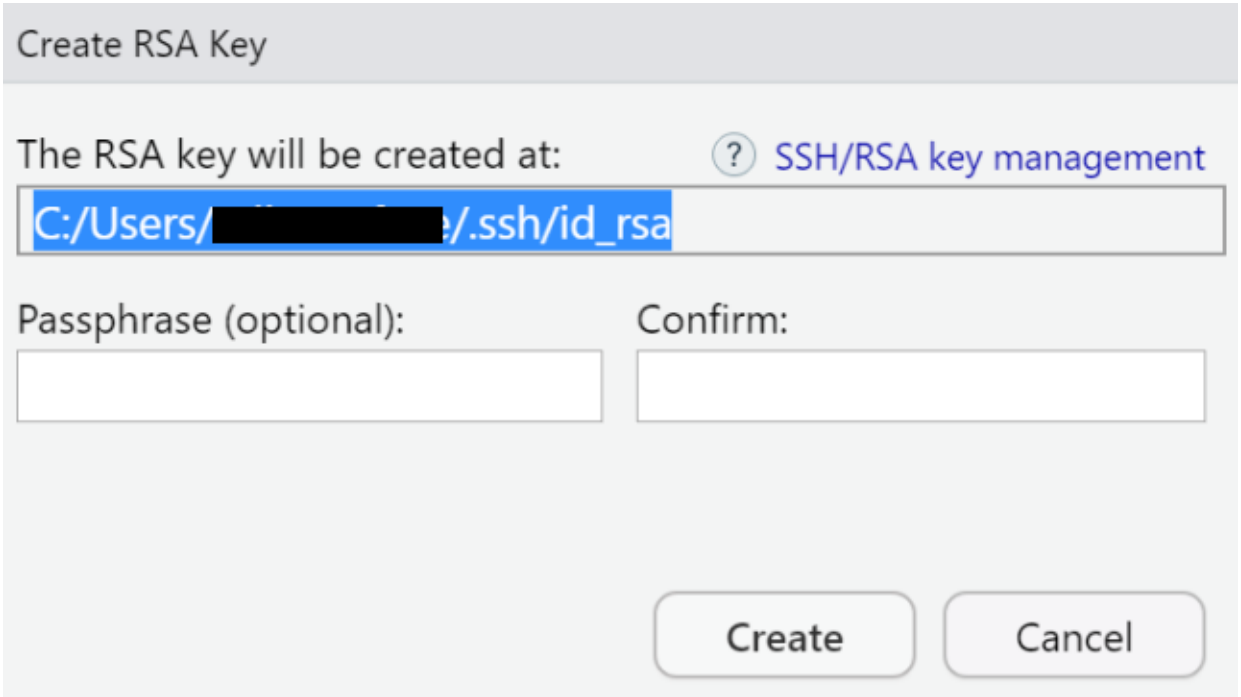
Open `Gitbash` (Windows users) or the Shell (Mac users), and configure `Git` with *your* username and email (**NOT MIKE DENLY!**):



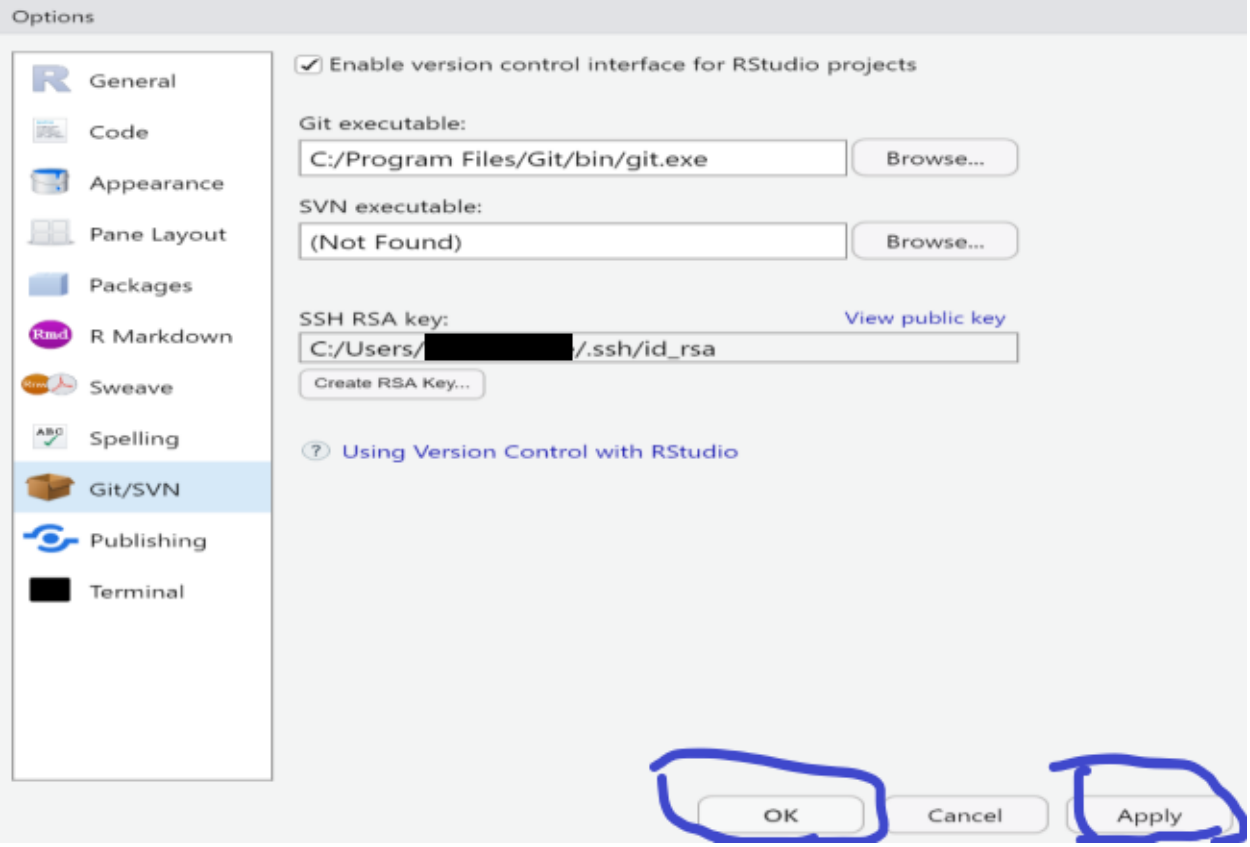
Then, setup an SSH key so that you don't have to type in your password every time that you do something on `Git`. Trust us that you really want to setup an SSH key, because entering in your password over and over again is not fun:







Don't forget to click "Apply" and "OK":



Then, ensure your SSH-key is enabled, and add it in `GitBash`:

Create a Github Account

After configuring `Git`, go to www.github.com and create a free account, using the same email address. (Note: you can also sign up for Github before, but just make everything is done with the same email address.) The free account is totally fine for our purposes. Because this account will be public and you will want to reference it later, it is generally better to create a username with your name in it. Bryan (2019) recommends that you keep your Github name the same as your Twitter name, etc. for consistency. That's probably a good idea. Additionally, you'll probably want to ensuring that the username is professional-sounding and is in all lower case.

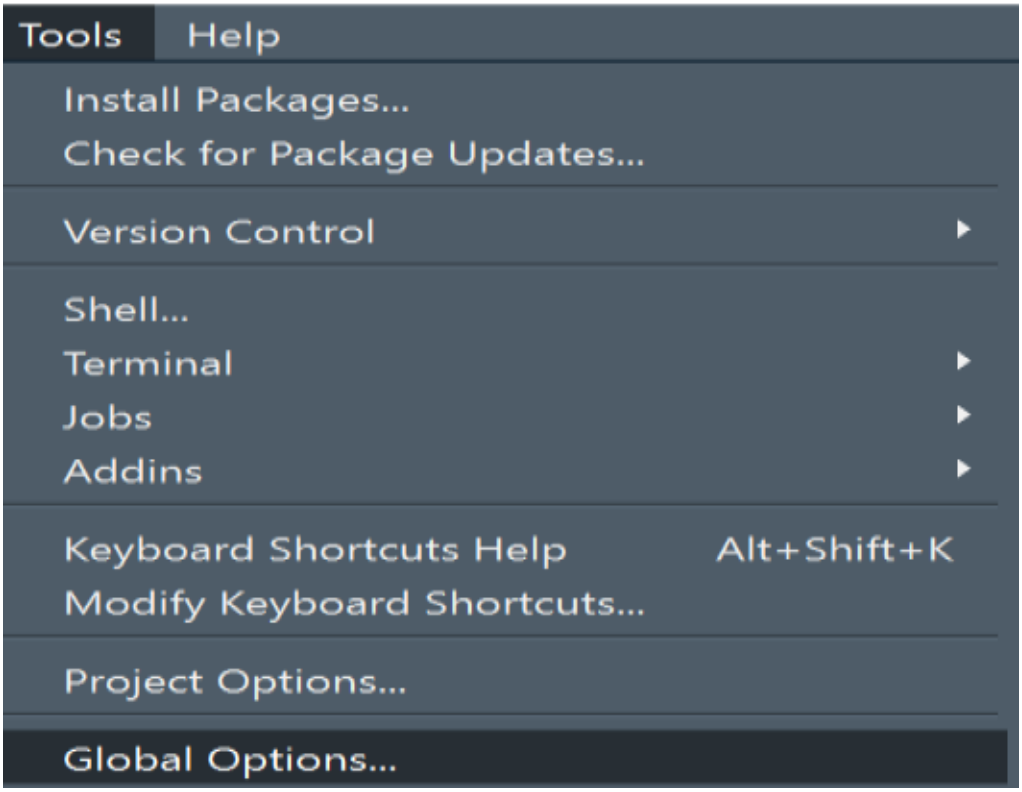
Github may ask you if you want to install their client software program. Most people don't

use the Github client, so we won't be using it in this class. Instead, they use `GitBash` (Windows users) or the Shell (Mac users).

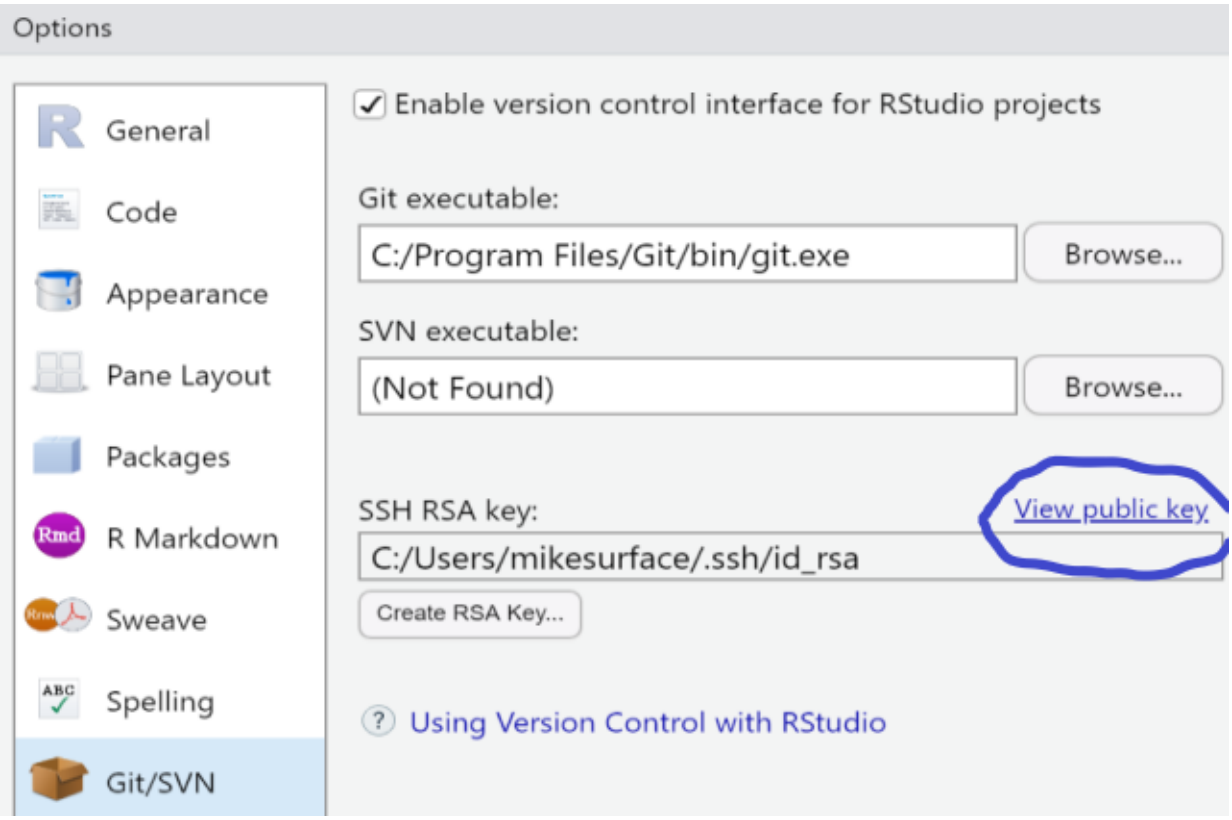
Associate Your SSH Key from `R` Studio with your Github Account

To avoid problems down the road, you will need to associate your `R` Studio SSH key with your GitHub account.

Recall that your SSH key is accessible under:



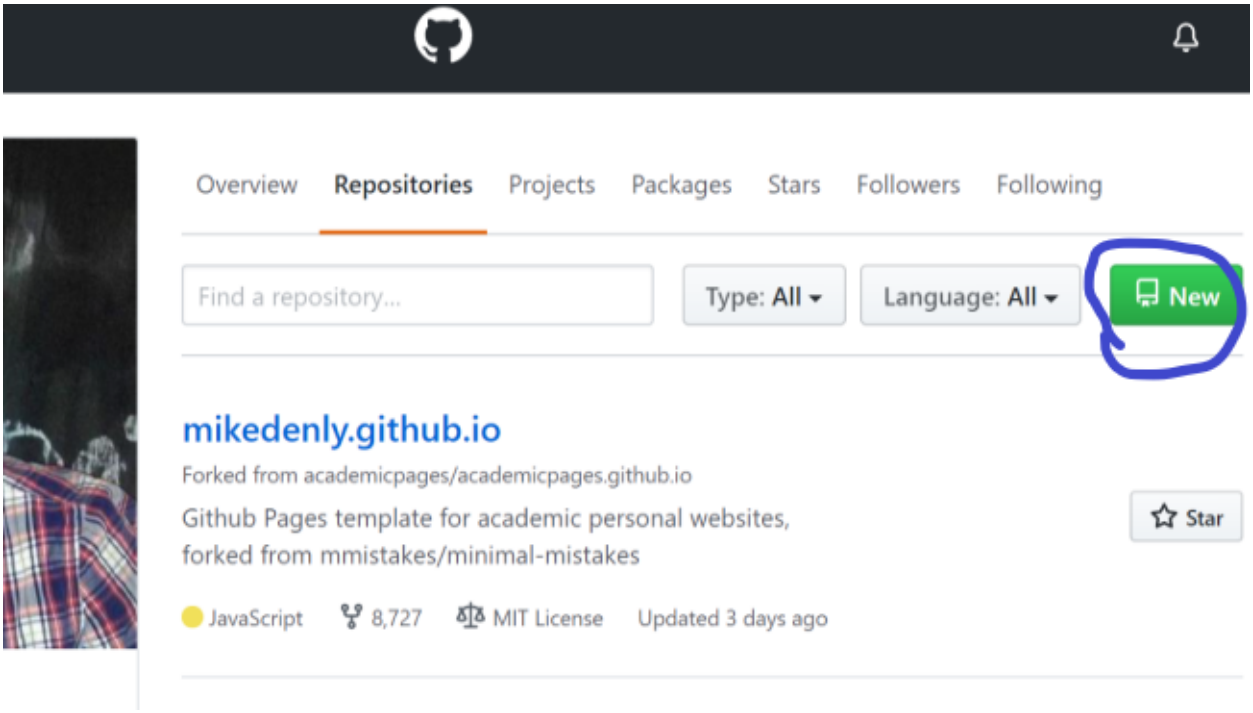
Then, click on "View Public Key", and copy it so you can paste it later:



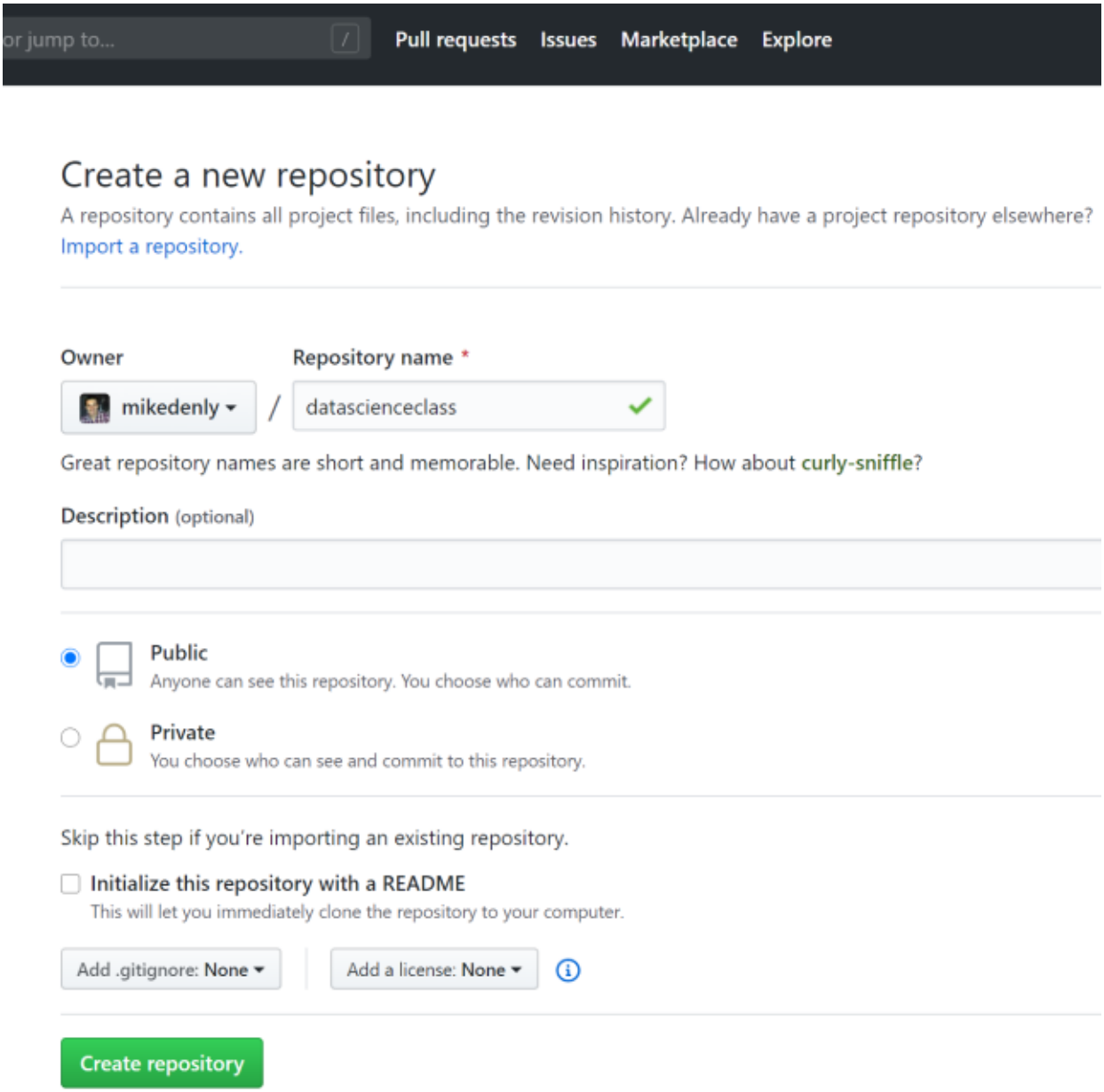
Follow this website to add your SSH key to Github. When doing so, I just called my key "R Studio".

Create a New Repository (repo) on GitHub and Initalize It

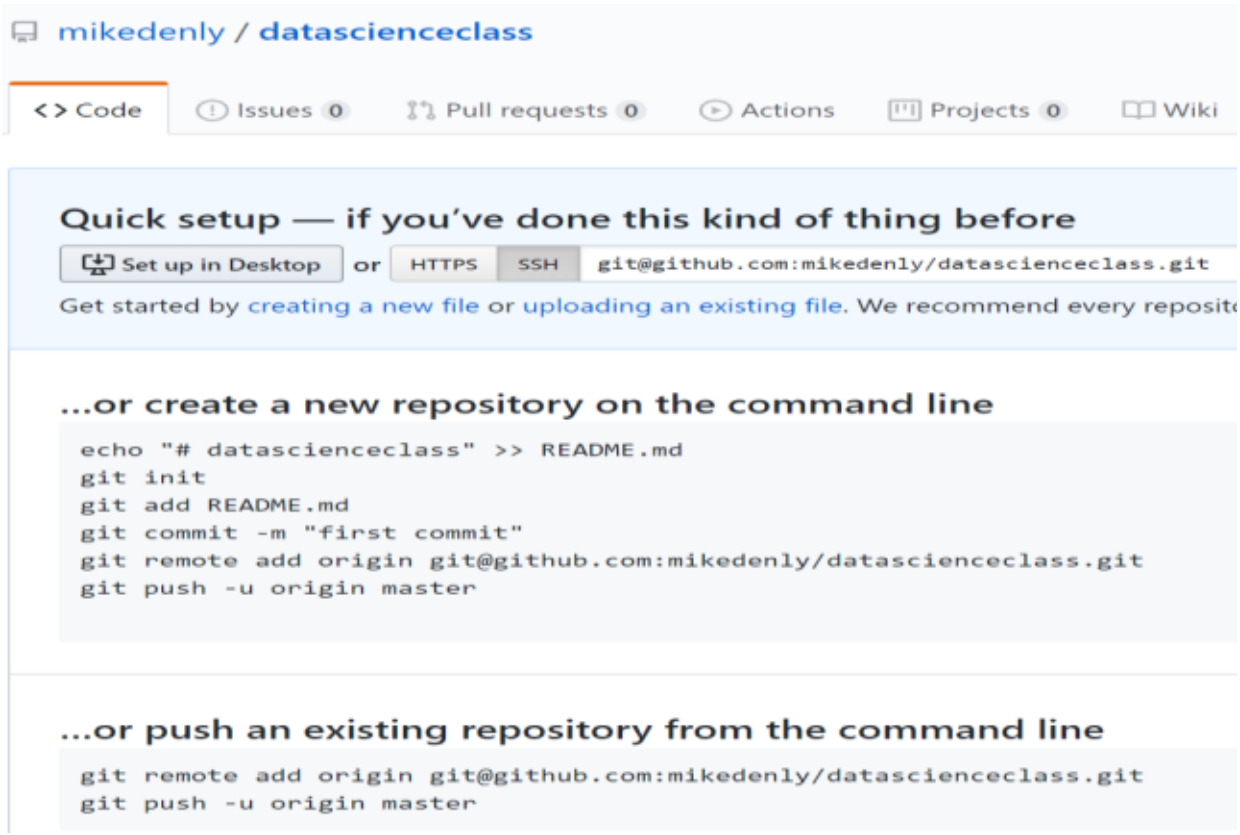
We will want to store our work in a repository on GitHub. From Github, click on "New" under the "Repositories" tab:



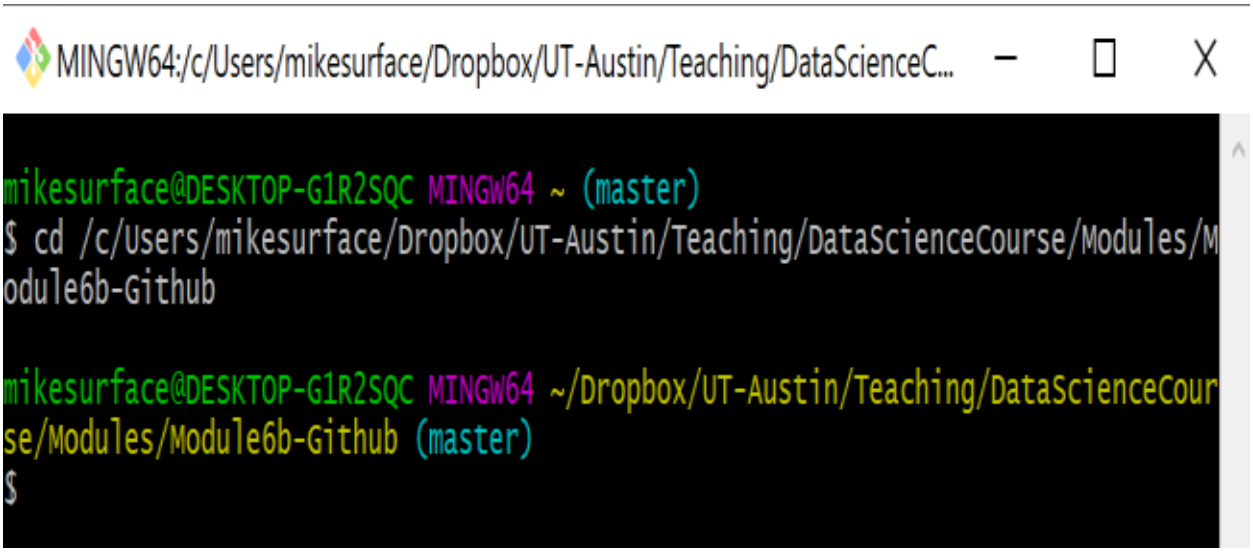
It will bring you to a page like the one below. I just called my repository "datascienceclass". Like with your username, avoid spaces, special characters, upper-case letters, etc.



After you create your repository, Github will provide you with some code to help you get your repository initalized:



Before running the bit of code, "...or create a new repository on the command line", you will first need to change your directory. To do so, go to `Git Bash` (Windows Users) or the Shell (Mac users), and change your directory with "cd" to the folder that you are using for these respective files.

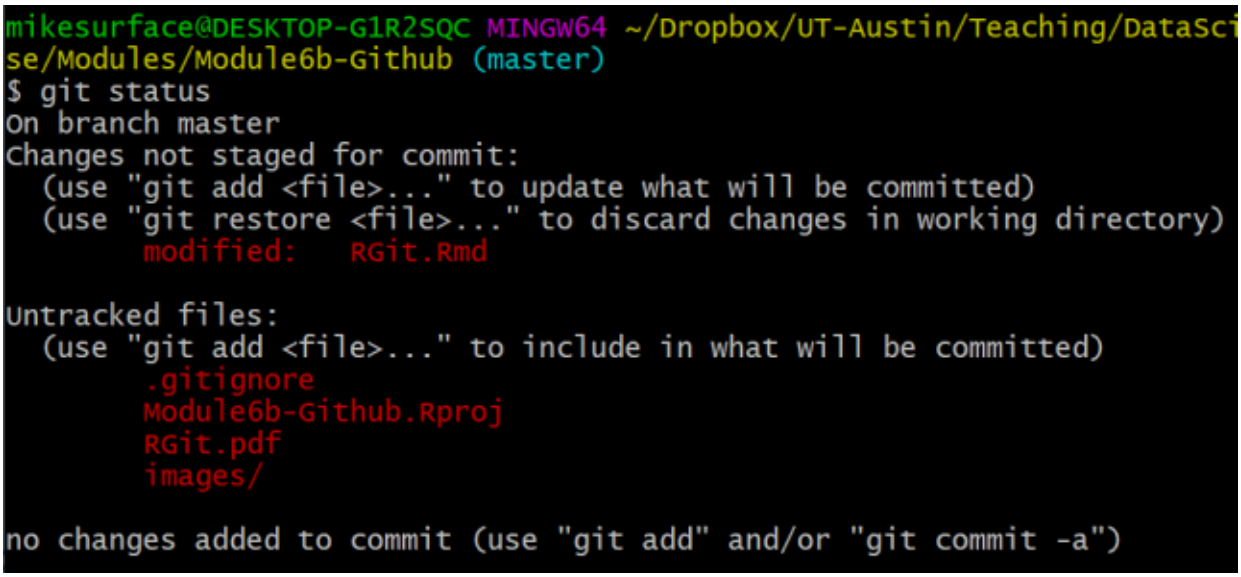


Then, run the above code suggested by GitHub in `Git Bash` (Windows Users) or the Shell (Mac users) to initialize your repository.

The Basic `Git` commands

You may be wondering what in the world those codes that you GitHub wanted you to run actually mean. Here are the very basic ones:

- `git init`
  - This command initializes a repository. Basically, you need to tell the folder that you are working out of on your computer that it is now going to be a `Git`-enabled.
- `git add`
  - This command allows you to add files to the staging area. The latter is the place that your files go before you can get that them onto Github.
- `git status`
  - This command allows you to check which files have been added to the staging area–that is, the place you need to put your files before getting them on GitHub. For example, the screenshot below is of my repository for creating this module. I had previously added the "Rgit.Rmd" file to the staging area, but then I modified it, which is why it is above and in red. If I had saved "Rgit.Rmd" and then immediately added it to the staging area, then it would have been in green. I never added the untracked files to the staging area, which is why they are where they are and are in red.



- `git commit -m "my descriptive message here"`
  - This command commits a file, which is the second-to-last step before putting it on Github. Whenever you commit file, you always add a message with `-m "message"` to indicate exactly what you are doing. If you forget to include a message, `Git` will not allow you to proceed. Generally, you should keep your messages short but descriptive. This way, you and others can follow exactly what you did at a later point. Generally, whenever we commit, we add `-a` to our command, making it: `git commit -a -m "my descriptive message here"`. Doing that allows you to commit more than one file at a time. Otherwise, you would need to add each

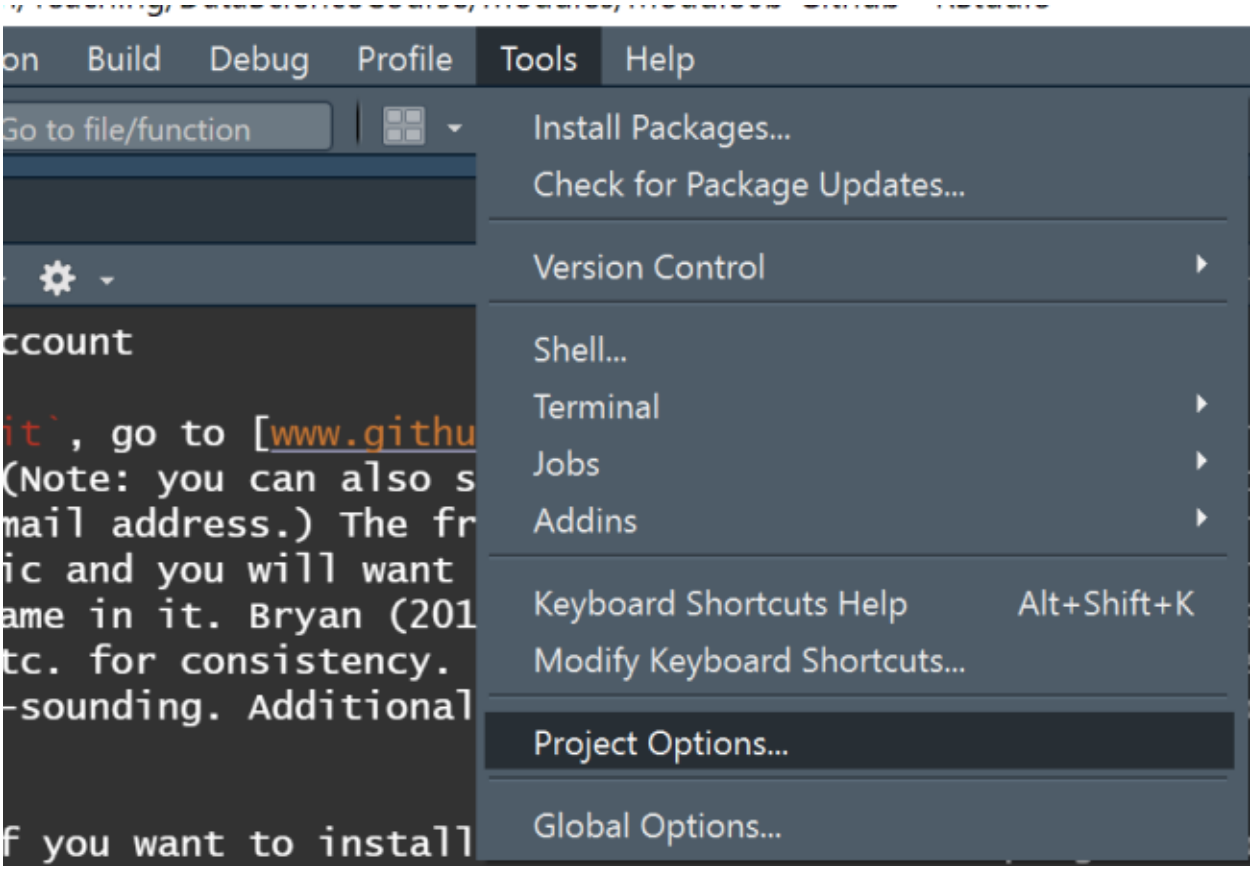
file individually, immediately commit them, and repeat. By adding the `-a` to our command, we can just add a bunch of files to the staging area, and commit them all with one step. If you use this course of action, though, make sure to keep your commit messages on point. If the files are not related, you probably don't want to commit them all at the same time, as that would make for very confusing commit messages.

- `git push`
  - This is the final command that you need to run in order to put the files on GitHub. Generally, you run `git push origin master`. What this means is that you are pushing the master (main) branch to GitHub. Naturally, your next question should be: what is a branch? And you may also be wondering: why in the world did I have to add the files to the staging area, commit them, and then push them? Don't these steps seem repeitive and unnecessary?

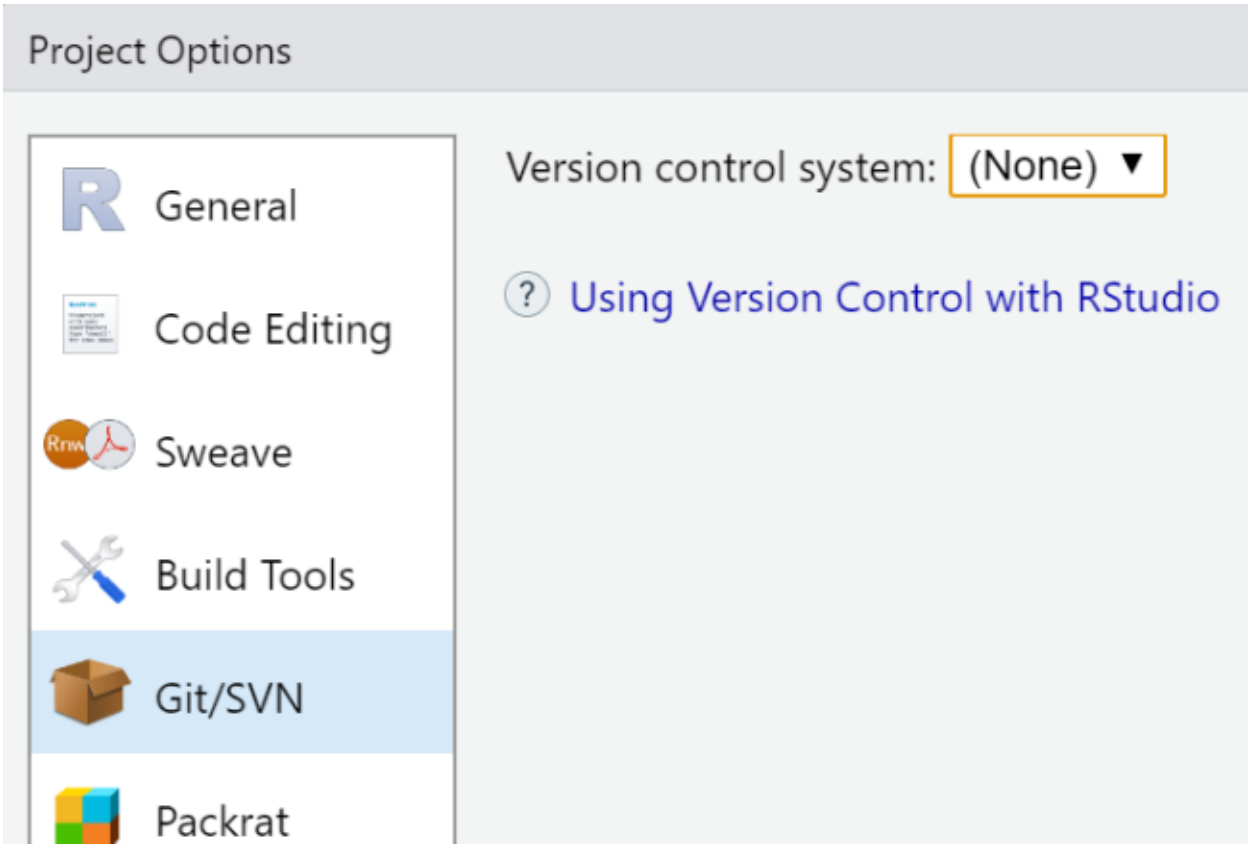
It turns out that these steps are all necessary, and being able to use more than one branch is another benefit of `Git`. Recall the earlier example with Denly, Findley, and Charm working together on a final project that involves statistical analysis in `R`. To make the example more concrete, let's say they were all struggling with figuring out the code to run a some very complicated model–let's call it the Ninja model. Each were working off the same file but on different branches to ensure nothing got overwritten. Then, Charm figured out a solution to the first part of the problem, so he told Denly and Findley. They then executed a `git pull` (i.e. the opposite of `git push`) of Charm's branch to their local computers, took a look, and agreed. So, Charm pushed his file to the master branch via `git push origin master`. However, they still weren't done yet. After hours of work, Denly figured out a solution to the second part of the Ninja model, so Charm and Findley pulled Denly's code to their computers, and took a look. Charm understood what Denly did right away. But it took Findley so long to figure out Denly's code that, by the time Findley finally finished, Denly and Charm were already done with their parts of the paper, sipping cold drinks at the beach without access to a computer. Accordingly, Findley had to push Denly's code to the master branch. At least, in the end, Findley made a tiny contribution something to the project :)

Creating a Repository (repo) in `R` Studio

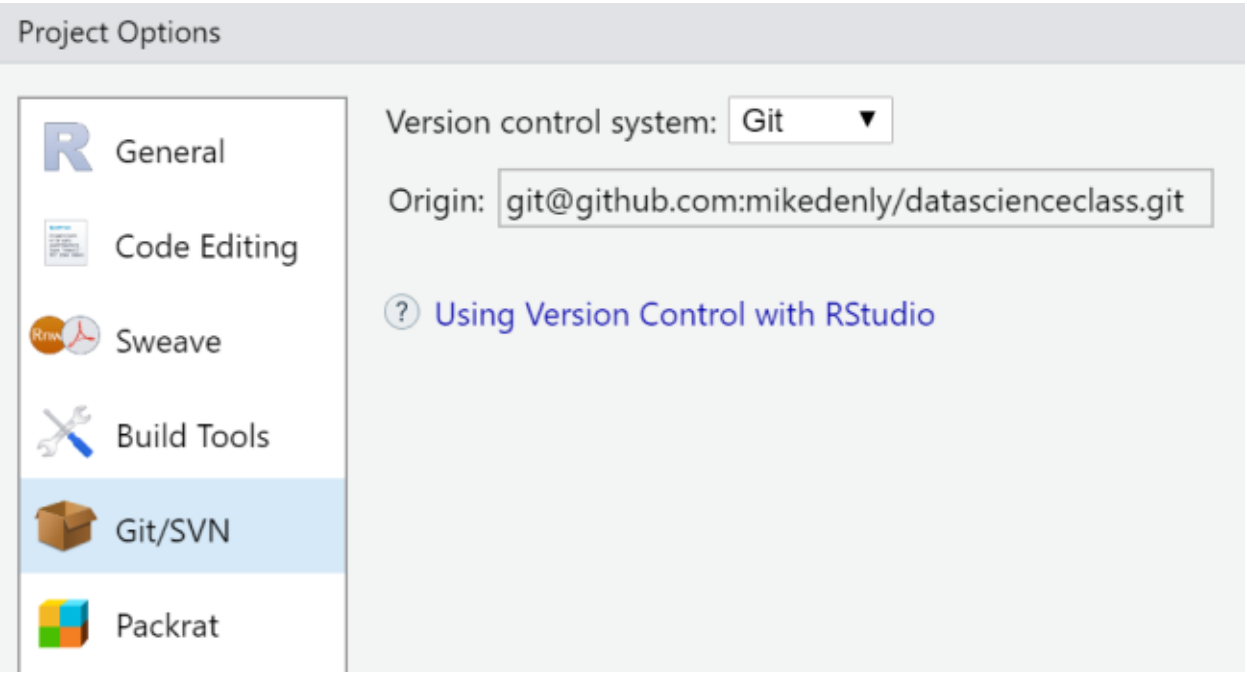
Now that we have `Git` set up, we know the basic of `Git`, and we have initialized a repository for our project with help Github, we need to create a repository in `R` Studio as well. To do so, go to:



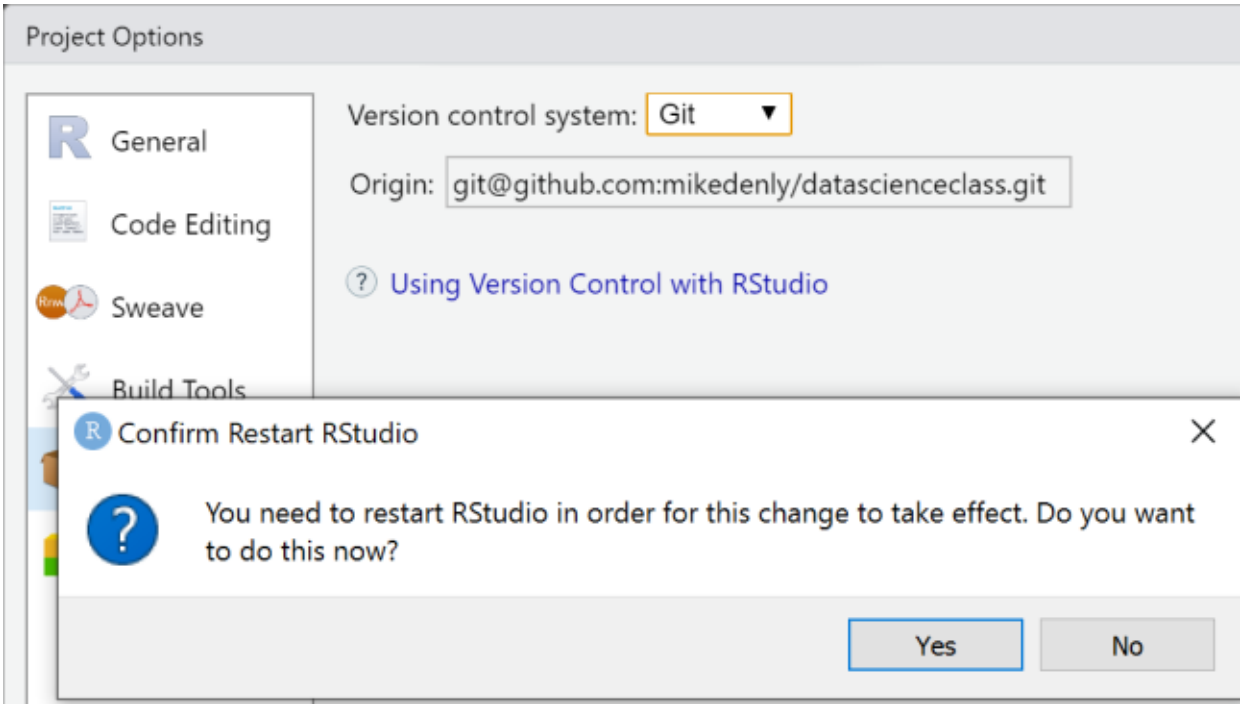
Change the version control system to `Git`:



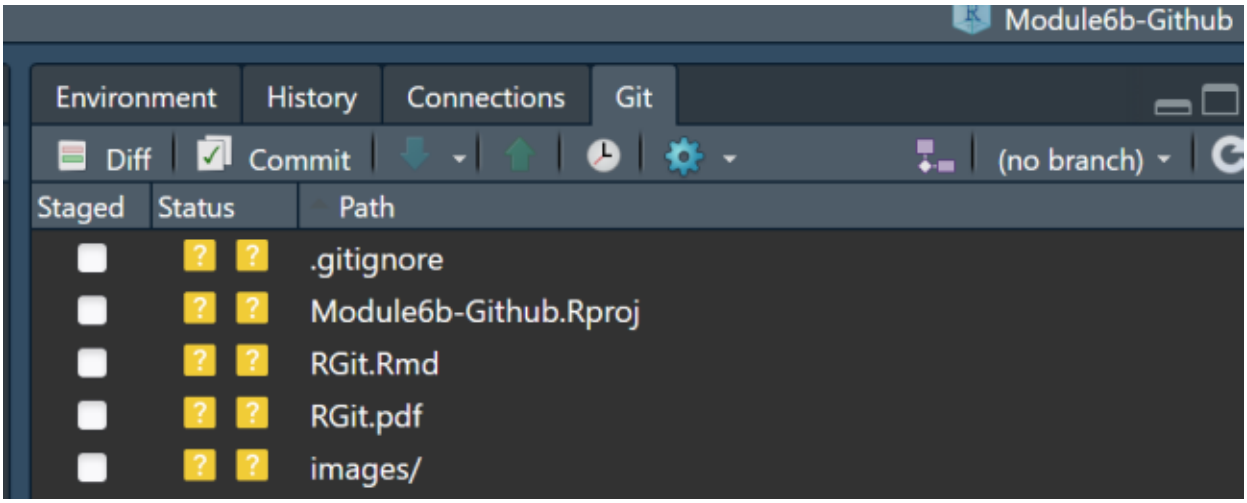
Since you already initialized your `Git` repo in a previous step (above), `R` Studio will know to change it to that `Git` repository. Note that if you want to the change repo that `R` Studio uses by default, you will need to initialize another repo in `Git Bash` (Windows users) or Shell (Mac users). Then, `R` Studio will use that new repo.



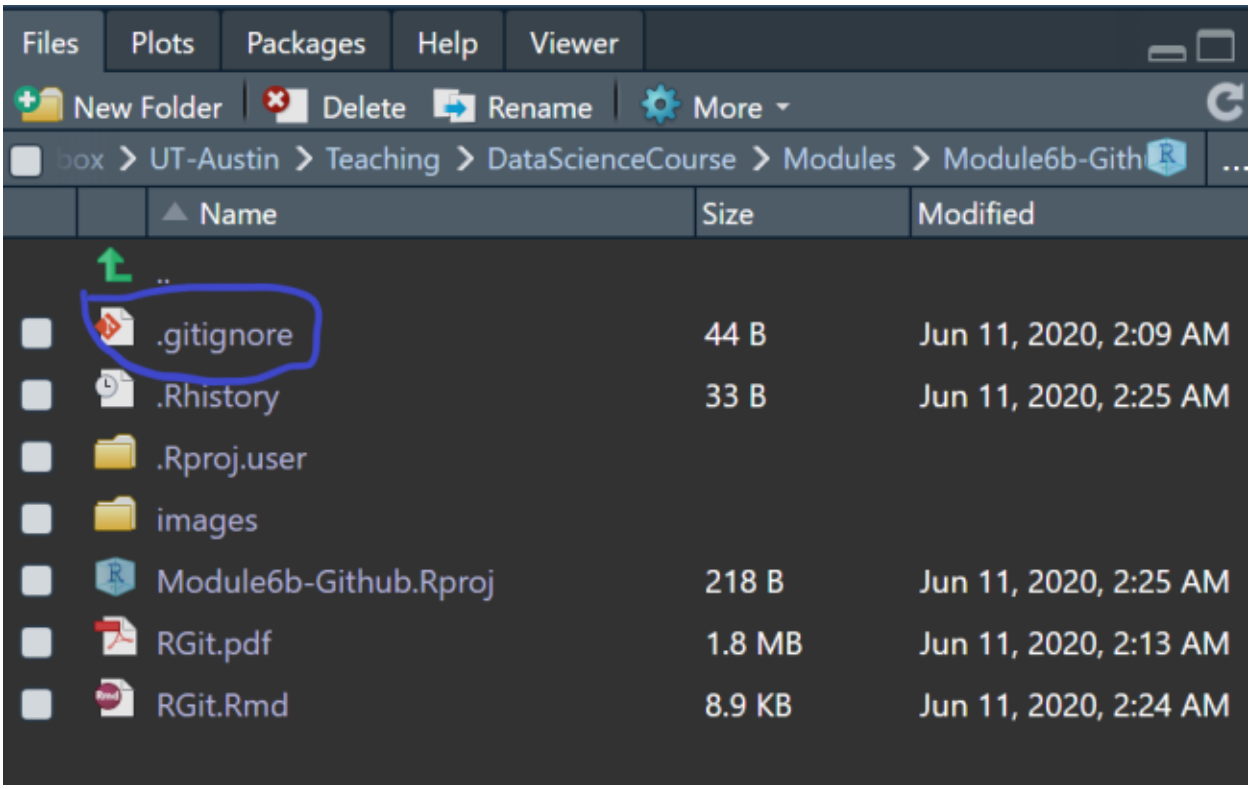
Restart your R Studio when it asks you:



After restarting, you will see a new pane in the upper-right portion of the screen that will allow you to run the basic `Git` commands:

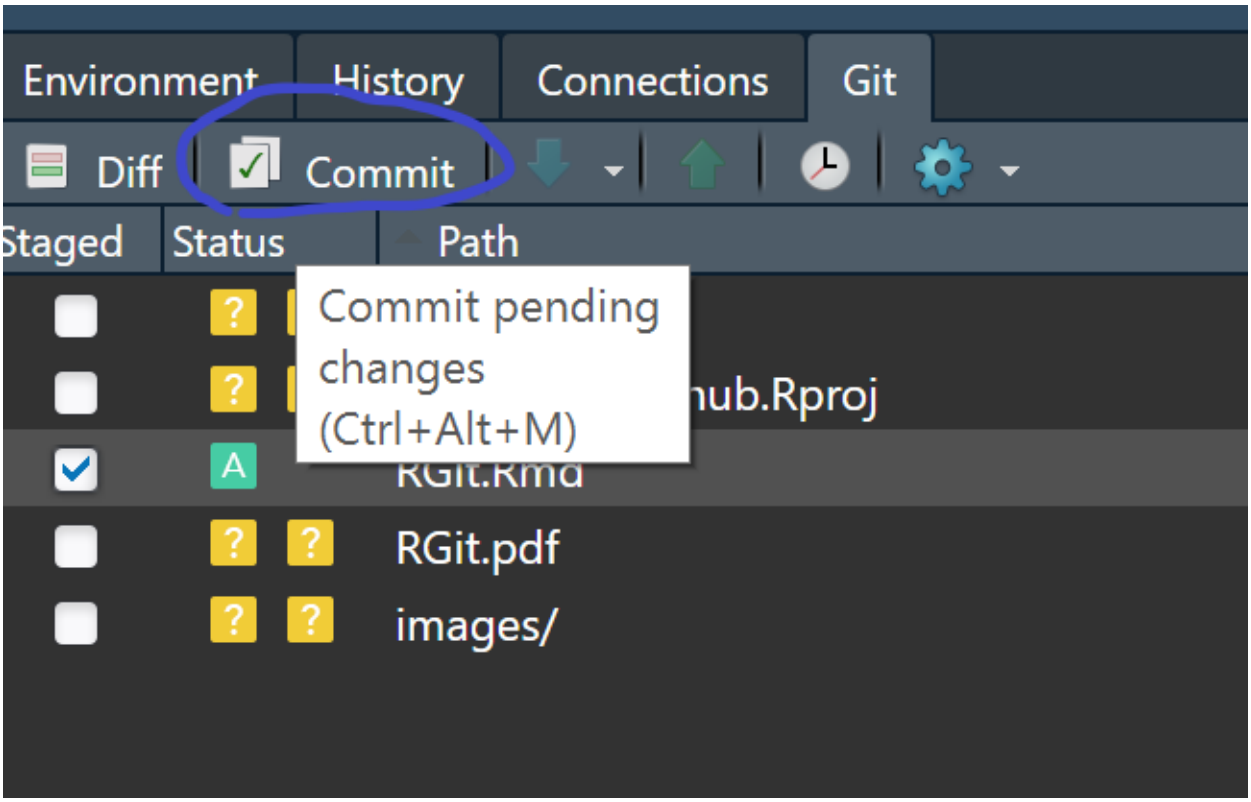


And you will notice a new ".gitignore" file in your directory:

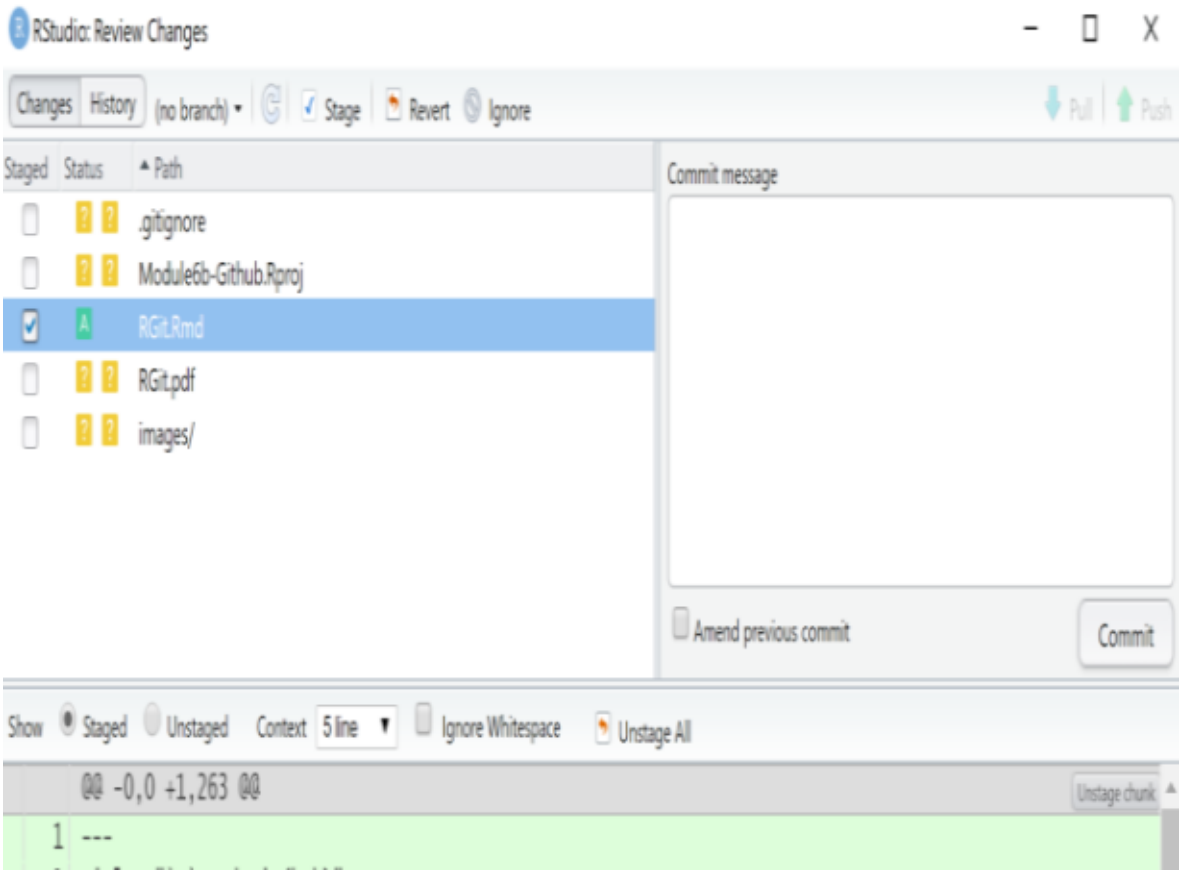


Using `R` Studio to Push Files to GitHub

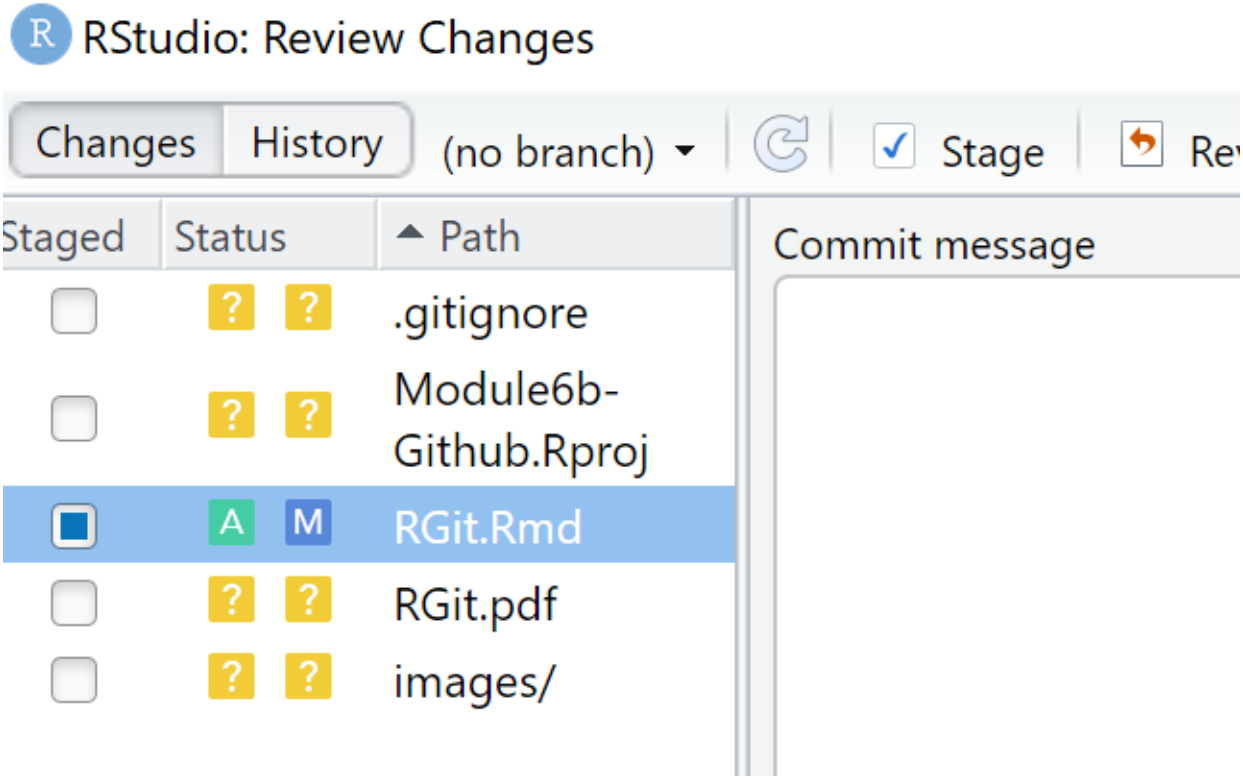
The `Git` pane in `R` Studio allows you to run `git add` and `git status` really easily. Notably, you just click on the "Staged" button to add something to the staging area. Files that aren't "Staged" are simply not in the staging area. To do the equivalent of `git commit`, you can just click on the "Commit" icon after adding your file to the staging area:



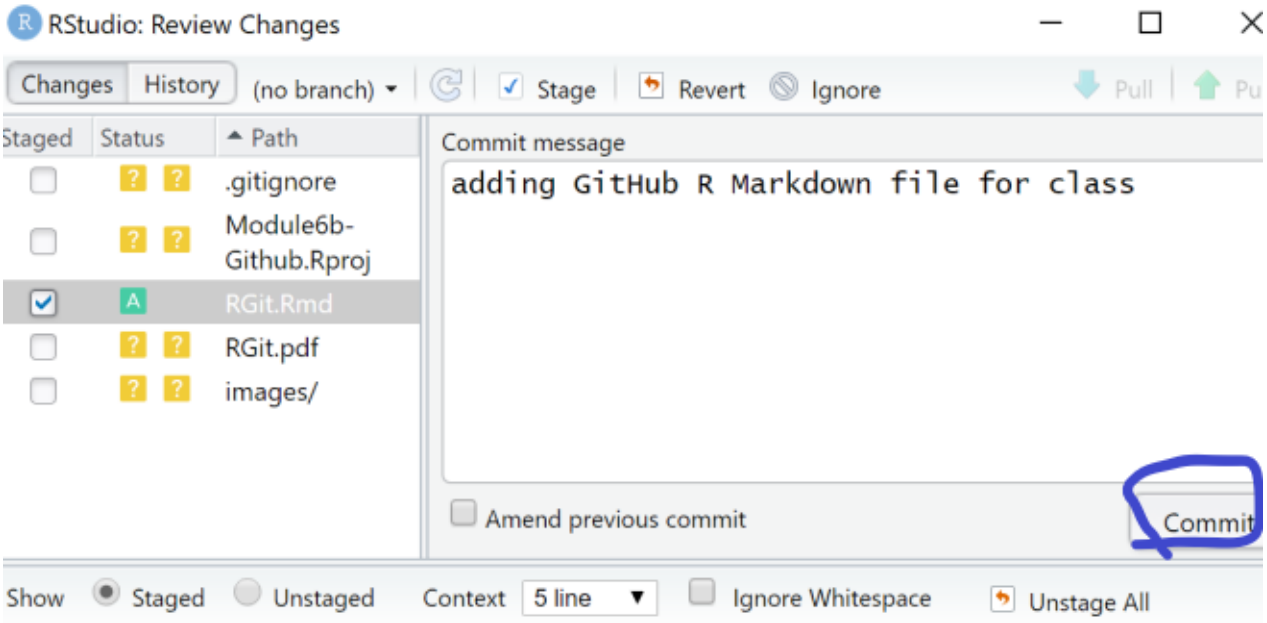
- After clicking on “Commit”, you will be brought to an area that looks like this:



Make sure to click "Staged" if you have modified and saved your file in the meantime:

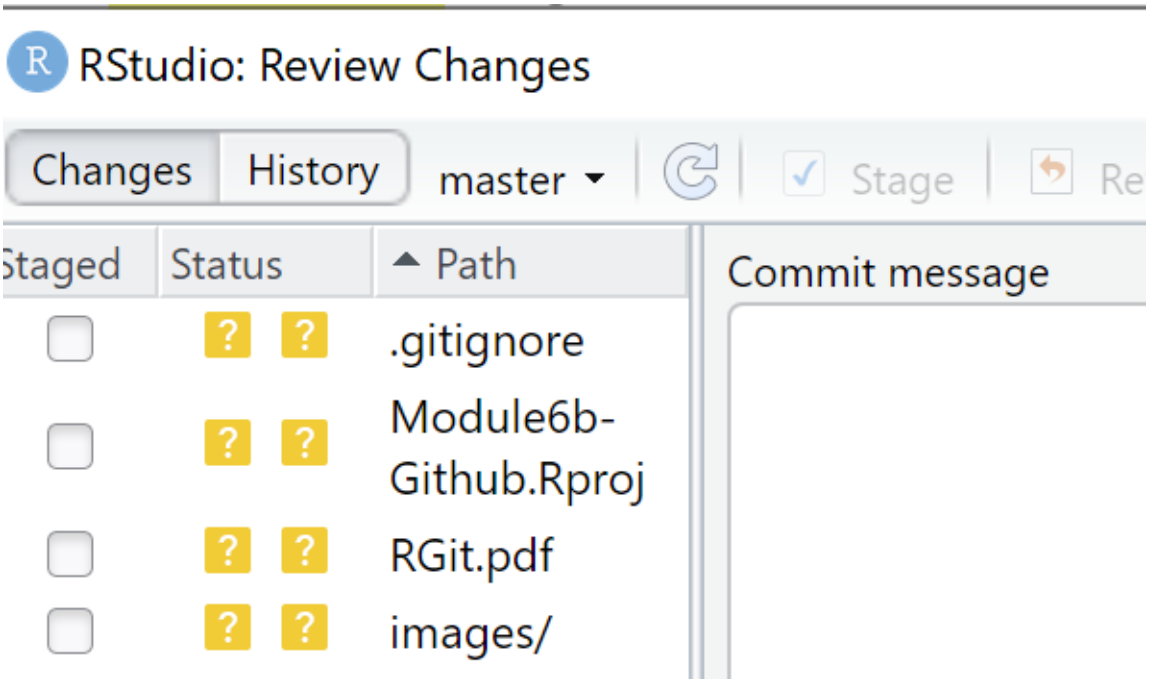


- Add your descriptive message in the top-right hand pane for what you did and click "Commit":

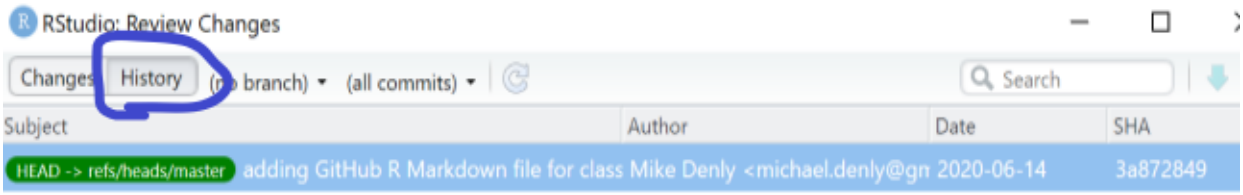


- You will then see something like this:

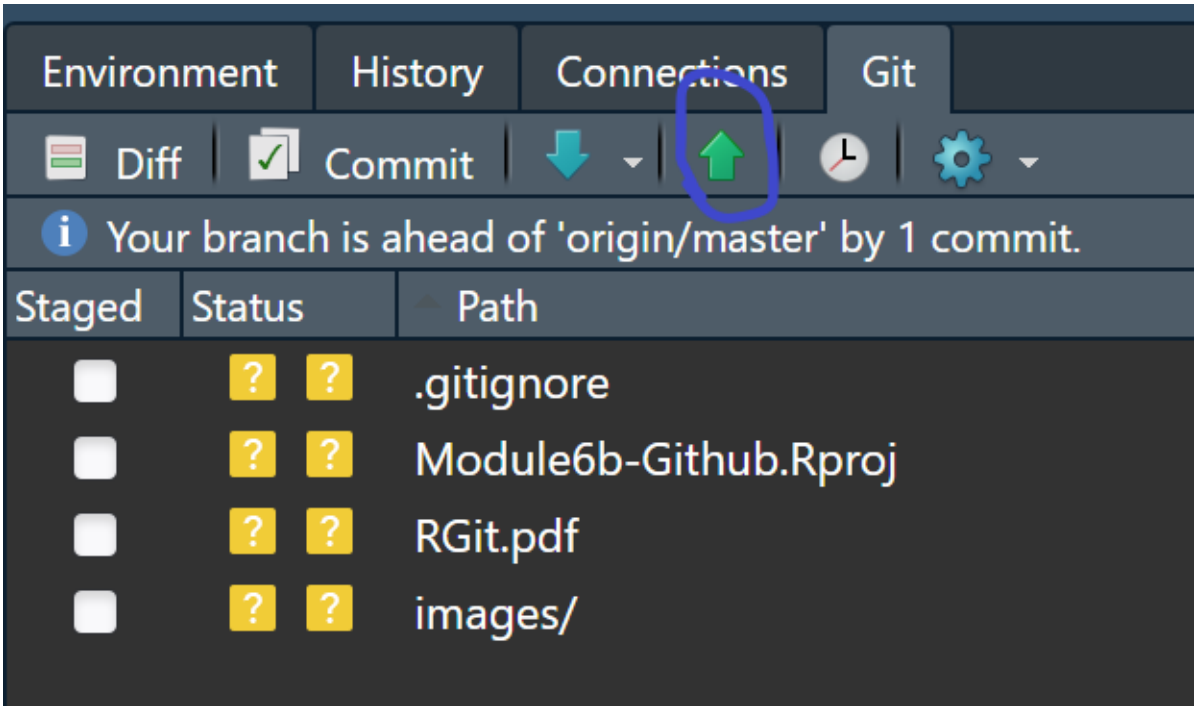
- After closing out, you will notice that the `RGit.rmd` file we were working has been committed and thus is no longer on the staging area:

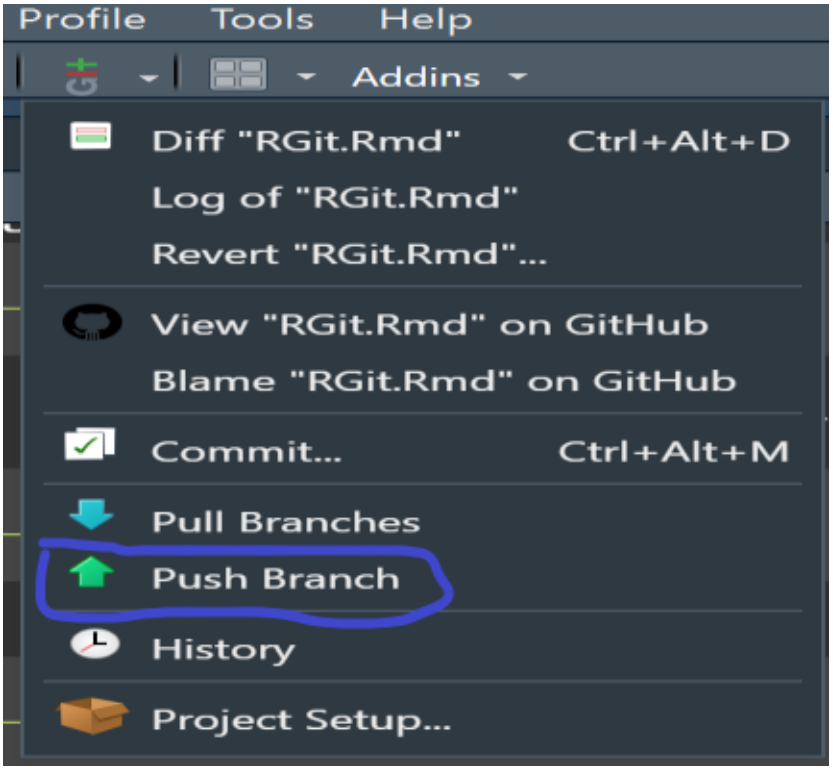


- You can also see the commitment history related to that particular file by clicking on the "History" tab:



- To push the file to Github, click on the Green up arrow or select "Push Branch" from the Git menu:





As should be clear, `R` Studio makes putting files on Github a little easier. However, you can still do things the old-fashioned way with running commands in `Git Bash` (Windows users) or the Shell (Mac users). Denly generally uses `Git Bash` for everything so that he does not forget all of the commands. Whichever way you choose is completely up to you.

References

Bryan, Jenny. 2019. "Happy Git with R."

Wickham, Hadley. 2019. "Git and Github."