Github Guide for Stat 495 Fall 2023

Much of this guide has been adapted from previous versions prepared for Stat 231 or Stat 495 by the Statistics faculty.

Expectations for students entering Stat 495

- Students should be able to use R/RMarkdown/RStudio. This can be using the R server or preferably, on their own machines. If you need instructions for installing on your own machine, please see the relevant section below. Again, for Stat 495, students are strongly encouraged to have their own installation of R/RStudio, rather than rely on the server.
- Students should be able to use Git/Github. This can be via RStudio (however implemented) or other avenues such as Github Desktop (see below if this is new to you and you'd prefer an app approach instead of working through R). If you need reminders about installing and setting up Git, etc. see the relevant sections below.

Instructions for 495

These instructions assume that you already have a Github account and access to R/RStudio. If not, check out those sections below, and then return here.

1. Create your own personal repository (*repo*) for Stat 495 course content under your Github username. Throughout the semester, you will copy files from the class repo to your own repo before making any edits and saving any changes (you will *commit* your changes and *push* them back to your repo). This should be done outside the class organization (though you'll join that to get course materials, details below).

a. Go to GitHub.com and click the **New** button to create a new repository. You can get to the same place by clicking the + dropdown menu in the top right and selecting **New repository**.

b. Create a new **private** github repo with an informative name that you will understand (e.g., "stat495", "stat495-content", "ada-content"). When you join the class organization, you will also have access to a separate repo for the course content I share with you ("stat495-f23-content"), be sure your repo's name is different and that you can tell them apart. While you can change your personal repo name after the fact, you should try to get it "right" the first time to avoid confusion later.

c. Make sure your repo is **private**, and check **Add a README file**. Later, you will be asked to put specific information in your README as part of Homework 0.

d. Click **Create repository**.

2. Follow these instructions to add me (amywagaman) as a collaborator on your personal repo for the course.

3. I will invite you to a private organization for this class (Stat 495 Fall 2023). Accept the invitation. You only have *pull* access to the *stat495-f23-content* class repo, meaning you cannot update the files in this repo (this is like "read" access versus "write" access).
4. In RStudio or in Github Desktop (your preference), open the class repo and your personal course repo. Be sure you can swap between the projects/repos.

5. At this point, you should be set up for the course. Need a refresher for the workflow moving between the class repo and your personal repo? Check below.

Reference
What are Git and GitHub? Why are we using them?
Getting Everything Set Up

- Set up a GitHub account
- Install Git on your local machine
- GitHub Desktop App or Configure Git within RStudio

Understand workflow for assignments with Git and RStudio
Hints and Tricks + Dos and Do Nots
Resources

What are Git and GitHub? Why are we using them?

Have you ever worked on a group project where sharing files with updates was a hassle? Or had issues where you've changed code and it broke but you can't remember its original working state? These things have happened to everyone. Git and GitHub are designed to try to minimize problems such as these by providing your projects with ***version control***.

So, what is version control? As described by Emily Robinson and Jacqueline Nolis in their book, Building a Career in Data Science, when describing skills data scientists need: "another core skill is using ***version control***—a method of keeping track of how code changes over time. Version control lets you store your files; revert them to a previous time; and see who changed what file, how, and when. This skill is extremely important for data science and software engineering because if someone accidentally changes a file that breaks your code, you want the ability to revert or see what changed.

Git is by far the most commonly used system for version control and is often used in conjunction with **GitHub**, a web-based hosting service for Git. Git allows you to save (*commit*) your changes, as well as see the whole history of the project and how it changed with each commit. If two people are working on the same file separately, Git makes sure that no one's work is ever accidentally deleted or overwritten. At many companies, especially those with strong engineering teams, you'll need to use Git if you want to share your code or put something into production."

Be aware though, that this is not quite the same as say, working in a Google doc, or coding in Google co-lab. Version control is useful even when working on your own (you can argue you are collaborating with yourself), but you can run into conflicts if folks are editing and committing the same files but are working in different places in the files. However, the version control will show you exactly what the differences between file versions are, and allow you to revert back to previous versions if things go awry.

Version control systems serve a critical role in helping individual analysts track their code and data in repositories and facilitate sharing of files with research teams. These systems allow users to maintain multiple versions of files with multiple users. GitHub is a wildly popular web-based interface to the flexible and powerful distributed Git revision control and file management system, with more than 24 million users and more than 67 million repositories. Git and GitHub interface with RStudio, allowing RStudio users access to their repositories without having to go to GitHub's web interface (though you may sometimes find it useful to view that too). Finally, if you are used to working with files and folders, having an app on your computer (GitHub Desktop) can provide a useful interface to Git and GitHub as well.

Getting Everything Set Up

Set Up a GitHub account

1. Register for an account on GitHub.com using your Amherst email address. We recommend choosing a username that incorporates your name (e.g., *katcorr*, *bebailey*, *amywagaman*). For advice on choosing a username and more information about GitHub Repos, see Chapter 4 of the *Happy Git and GitHub for the useR* online book.

Install Git and R/RStudio on your local machine

1. If you haven't already, install R, RStudio, and TeX. See Kitty Girjau's Intro Stat Technical Guide for detailed installation instructions. **Even if you already have these installed, you should update to the latest versions** (this may involve reinstalling and/or updating R packages as well, including tinytex).

2. Install Git. Directions for both Windows & Mac are available in Chapter 6 of the *Happy Git and GitHub for the useR* online book. Windows users should follow Option 1 in Section 6.2. Mac users can follow Option 1 in Section 6.3 if comfortable, otherwise follow Option 2.

Github Desktop App or Configure Git within RStudio

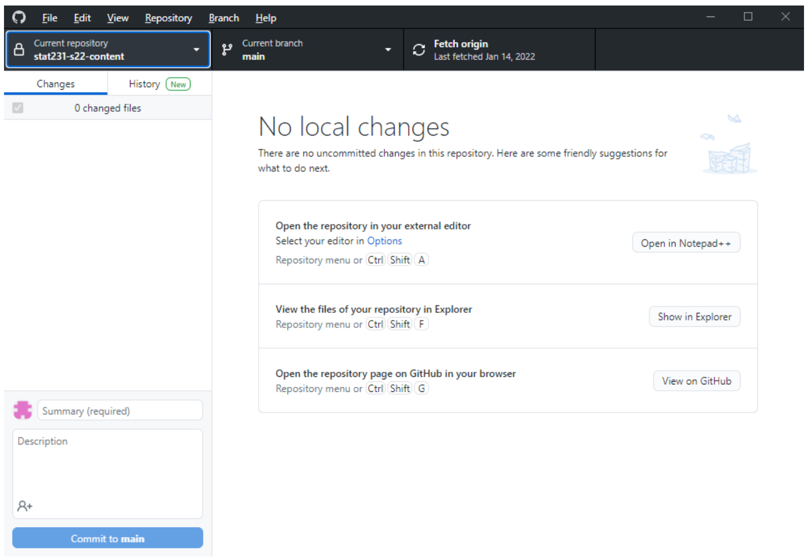
Github Desktop App

Are you more comfortable with files and folders than the RStudio interface? Do you prefer working with your Finder or File Explorer than in R? Do you expect to use Git with more than R code/files? Maybe you work with Python code a lot or you want to use it for Word documents? Would you prefer working with an app rather than through the RStudio interface?

If you answered yes to any of these questions, you should check out the GitHub Desktop app, rather than trying to configure Git within RStudio.

The GitHub Desktop app functions the same way as the RStudio interface does (if you saw this in a previous class), but allows you to view the files as files in folders on your computer. If you are more comfortable moving or working with files this way, it's a good way to proceed.

Here is an example of me looking at a Stat 231 course content repo in the app, and then using Repository > Show in Explorer to see the folders and files. You can also click to view the repo online on GitHub.com. The app is available for both Windows and Macs.



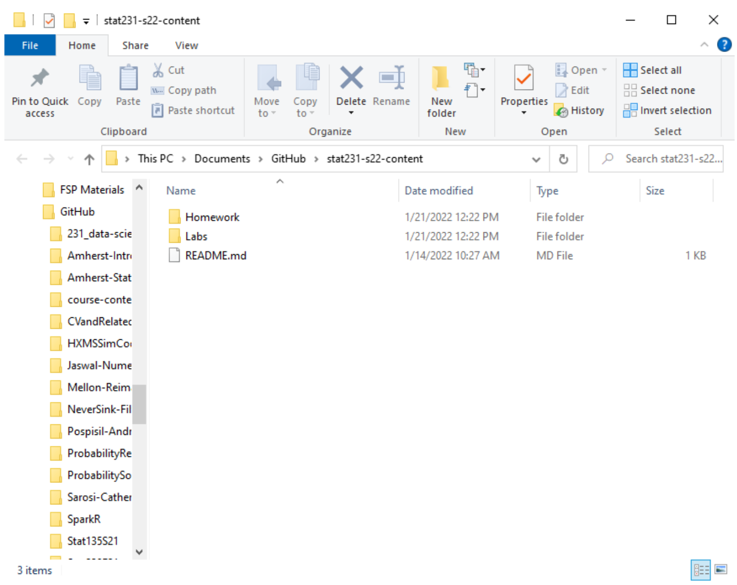
No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl Shift A
Open in Notepad++

View the files of your repository in Explorer
Repository menu or Ctrl Shift F
Show in Explorer

Open the repository page on GitHub in your browser
Repository menu or Ctrl Shift G
View on GitHub



Name	Date modified	Type	Size
Homework	1/21/2022 12:22 PM	File folder	
Labs	1/21/2022 12:22 PM	File folder	
README.md	1/14/2022 10:27 AM	MD File	1 KB

DANGER: If you use the app, remember that you have to go into it and pull, commit, and push files as needed. It will be outside the RStudio interface, so you have to remember to do that. In the past, I have sometimes worked on a file at home, then forgot to commit and push (or at least, push) and expected to have the updated version at my office. It won't be there unless you commit and push.

Why might you still like the app? Personally, I much prefer working with files and folders in this interface than within R, especially if I'm trying to move things around or re-organize or am not working on R code at the time. If you'd rather use Git within RStudio, those instructions are next.

Configure Git within RStudio

If you choose not to use Github Desktop, you'll need to configure Git within RStudio. (If you use Github Desktop, you should skip to the next section). That means that after completing the GitHub account and repo setup, and installing Git on your local machine, you should walk through these steps *both on your local machine and on the RStudio Server* (as a back-up in case you cannot get RStudio to work on your computer).

1. In the **console** in RStudio, use the **usethis** package and the code below to set your name and email, substituting your name (or GitHub username) and the email associated with your GitHub account:

```
## install if needed (do this exactly once):
## install.packages("usethis")
library(usethis)
use_git_config(user.name = "Jane Doe", user.email = "jane@example.org")
```

Alternatively, the setup options can be configured via the shell in RStudio. Go to the **Tools** menu, and select **Shell. . .** , and then run the shell commands below, substituting your name (or username) and the email address associated with your GitHub account. Close the shell when you are done.

```
git config --global user.name 'Jane Doe'
git config --global user.email 'jane@example.com'
git config --global --list
```

2. Generate *ssh* (secure shell) keys so you don't have to keep authenticating your credentials with GitHub. *You only need to do this once per device*:
a. Go to the **Tools** menu, and select **Global Options. . .**
b. Click on the **Git/SVN** option, and click the **Create RSA Key. . .** button.
c. In the prompt, just click **Create**. Do not set a passphrase, otherwise you will be prompted regularly for your password.
d. Close the next prompt that pops up with the RSA key information.

3. Add public keys to github. *You only need to do this once per device*:
a. Still in the **Git/SVN** options (**Tools** ! **Global Options. . .** ! **Git/SVN**), select the **View public key** link, and copy the key from the window that pops up. If the link isn't there, you may need to close and re-open RStudio for the link to show (or start a new session on the RStudio server).
b. Go to GitHub.com, sign in, and click on your profile picture icon in the top right.
c. Select **Settings** in the dropdown menu, then click on **SSH and GPG keys** in the menu on the left side of the settings page.
d. Click the **New SSH key** button, and paste the key into the appropriate space. Give the key a relevant title, e.g. *Amherst RStudio Server*.
e. Click **Add SSH key** when you are done. GitHub might prompt you for your password after (you don't want people adding in keys without your knowledge!), and should

email you regardless to let you know a new key has been added to your account.

4. Clone the class repo in RStudio. *You only need to do this once per device*:
a. Open the class content repo on our class's GitHub page, https://github.com/stat495-f23.
b. Click on the **Code** button , and select **Use SSH** (unless you've used HTTPS before and are familiar with how it works).
c. Click on the **copy to clipboard** button to the right of the textbox to copy the URL.
d. Back in RStudio, click on **File > New Project. . . > Version Control > Git**.
e. Paste the repository URL in the first textbox.
f. Make sure the project directory name in the second textbox is "stat495-f23-content" (to help distinguish the course repo from your own).
g. For the textbox labeled *Create project as a subdirectory of:*, choose or create an appropriate location for the repo. There are a couple philosophies around organizing github repos:
• **Option 1**: *Organize all your github repos in one place*. On the RStudio server, for example, change the text from *"~"* (your home folder) to *"~/git"* by clicking "Browse" then creating a *git* folder as a place to put all of your github repos.
• **Option 2**: *Organize your github repos by course or major project.* One the RStudio server, for example, change the text from *"~"* (your home folder) to *"~/stat495"* by clicking "Browse" then creating a *stat495* folder as a place to put all of your repos for this course.
h. Click on **Create Project**. The first time you connect to GitHub from within RStudio, it will prompt you with a warning message saying "*The authenticity of host github.com... can't be established. . . .*" You should type "yes" into the box to trust this host and submit.

5. Repeat Step 4 to clone your personal repo within RStudio (*also once per device*).

Understand workflow for assignments with Git and RStudio

This process will be the same whether you are on your own machine or the RStudio server, and also whether you are using the Github Desktop app or working with Git inside RStudio.

Everyone in the class has access to the *stat495-f23-content* repo. This repository will be updated throughout the semester, and it will allow you to have the most up-to-date materials on your local computer (or server profile if using the RStudio server).

You should have already cloned the *stat495-f23-content* repo and your own personal repo for the course.

1. Navigate to the course repo either in Github Desktop or in RStudio (depending on your setup).

2. Pull/Fetch any changes to the repo. Remember, you don't have permission to push any changes to the *stat495-f23-content* repo, so you'll just be pulling/fetching from here. If you get an error about conflict, don't freak out! This can happen if you accidentally edit one of the files in the *stat495-f23-content* repo – you can't push it back though, so just fetch/pull or reclone the repo if needed.

3. Copy the relevant file to your own repo.

4. Open the relevant files from within your repo to work on them.

5. When ready, commit the changes you made to the file. Making commits keeps a record of what changes you make and when. Try to use informative commit messages.

(Note: In the Github Desktop App, once you've made a commit, the "Fetch" option changes to "Push" so you know that's available. It won't let you "Fetch" again on the active repo until you "Push" what you've done. You can have multiple commits before Pushing though.)

6. When ready, either after your commit or several commits (for longer working sessions), Push. Pushing sends the current file version (from the last commit) to the web so others can see what you've done, including the commit log (and prior un-pushed commits).

At this point, you could go to GitHub.com and verify that the new file is in your own repo under the appropriate folder.

What to know about committing:

- You should commit somewhat frequently. At a minimum, I recommend if you are working on a homework assignment, you make a commit each time you've finished a question. Also, I recommend not starting to work on another project/assignment, etc. without committing where you ended up.
- Leave brief but informative commit messages that summarize the "why" of the commit with just enough detail that a collaborator (or yourself several weeks, months, or years

down the road) can make sense of it. For example, "Add plot" is not an informative message, but "Add scatterplot of housing prices vs square footage in Hampshire county to finish Problem 1.3 of Homework 3" provides way too much detail. "Add housing scatterplot in HW3, 1.3" might be a happy medium.

- To get you started with good habits, one-line commit messages should start with a capital letter, should not end with any punctuation, and should complete the sentence, "If applied, this commit will *commit message....*"

Summary

1. Pull/fetch to get changes that the instructor, yourself, and others (when working in teams) have made to the repos.
2. Copy files from the class content repo to your repo to work. Be sure you're in the repo you want to be in when working (usually your personal repo).
3. Commit document changes you have made to the code.
4. Push to send your commits (your changes) to the server so others can pull them.

Hints and Tricks + Dos and Do Nots

1. Pull often!
2. Conflicts (between files) will happen as you learn the system. Don't fret. Assuming you pull, commit, and push regularly, you are not likely to have major issues.
3. Version control is NOT the same as Google docs or Google co-lab. You don't want people working on the same file at the same time. But it is ideal for people working on a project at different times who need to share files.
4. You don't have to add ALL files generated as you work to Git immediately. For example, you can choose not add pdfs (or other files that are generated by your Rmd) to GitHub until you are ready to submit an assignment. Save the code though!
5. When submitting "final" files, such as an assignment pdf, consider giving the pdf a unique name. For example, if your file is *lab01.Rmd* (from the class repo) which generates *lab01.pdf*, rename this file as *YourName_lab01.pdf* before pushing (including a date may be useful too).
6. Do not add large files (50mb or larger) to GitHub. Things will break. You will get a warning when you try to commit these. If you commit accidentally and try to push, it will fail. Undoing the commit can be annoying (sometimes, impossible), so again, do not add large files to GitHub.
7. If you get stuck or run into a conflict when pushing or pulling, try running git status at the shell (**Tools** > **Shell**).
8. Really stuck? Please contact me or ask a peer for assistance. Practice your troubleshooting skills together.
9. Stay organized. The class content repo is set up a certain way for a reason – to keep files organized. You can use this structure or another structure that works for you. However, I should be able to easily find a requested file in your repo without sorting through everything. That is to say, you shouldn't keep every file in the main folder of the repo.

Resources

Robinson, Emily, and Jacqueline Nolis. *Build a Career in Data Science*. Manning Publications, 2020. https://livebook.manning.com/book/build-a-career-in-data-science/part-1/

Happy Git and GitHub for the useR by Jenny Bryan and Jim Hester

RStudio, Git, and GitHub section of *R Packages* book by Hadley Wickham and Jenny Bryan

Learn Git Branching interactive learning guide for Git (While we are not using branching, it can be useful to understand. Simply, it is a way for everyone to have their own “section” of a repo, with a “main” branch. Branches could contain different versions of content from the main branch. All your commits in our class will be to the main branch.)

GitHub Guides

Last edited: August 23, 2023 by Amy Wagaman