GitHub Classroom Guide*

Contents

What is GitHub and why are we using it?

In their popular book, Building a Career in Data Science,[1] Emily Robinson and Jacqueline Nolis write about core skills of a data scientist:

> "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."

The American Statistical Association's 2014 *Guidelines for Undergraduate Programs in Statistics* reinforced the need for students to be able to undertake increasingly complex data analysis using sophisticated approaches. According to the guidelines, students should undertake these analyses in a "well-documented and reproducible way." In addition, statistics is increasingly a team sport,

*Much of this guide is taken from guides developed by Nick Horton and Kat Correia.

[1]Robinson, E., & Nolis, J. (2020). Build a Career in Data Science. Manning Publications.

and students need to "demonstrate ability to collaborate in teams and to organize and manage projects."

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.

1. Set up GitHub account and repos

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*, *kdonges*). 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.

2. Create your own personal repository (*repo*) for the course content under your 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).

   a. Go to GitHub.com and click the **New** button to create a new repository. You can get 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 (e.g., "stat231", "stat231-content", "ds-content"). You will eventually have separate repos for the course content I share with you ("profbailey-course-content"), and for each of your three major projects (calendar query, shiny app, and blog). You can change the repo name after the fact, but you want to try to get it "right" the first time to avoid confusion later.

   c. Make sure your repo is **private**, and I encourage you to check **Add a README file**.

   d. Click **Create repository**.

3. Follow these instructions to add me (`bebailey`) as a collaborator on your personal repo for the course.

4. I will invite you to a private organization for this class (*STAT 231 - Data Science - Fall 2021*). Accept the invitation. You only have pull access to the *profbailey-course-content* class repo, meaning you cannot update the files in this repo (this is like "read" access versus "write" access).

2. Install GitHub 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.

3. Configure Git within RStudio.

After completing the GitHub account and repo setup and installing GitHub 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")
```

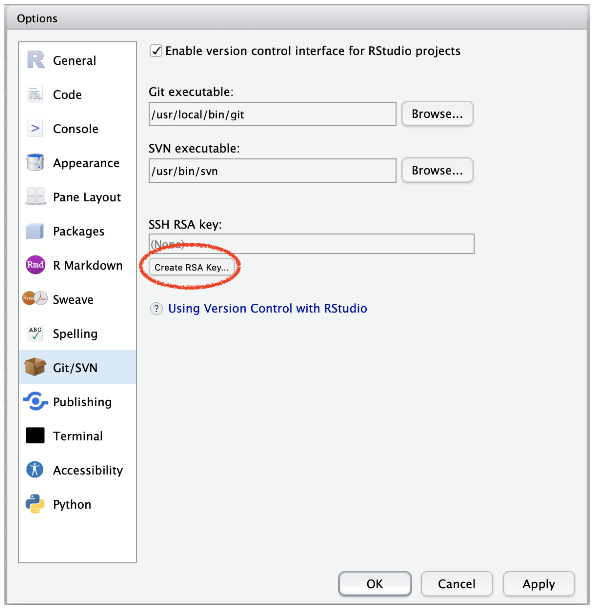
   Alternatively, the setup options can be configured via the shell in RStudio. Go to the **Tools** menu, and select **Shell. . .** , and then run the shell commands below, substituting your name (or username) and the email address associated with your GitHub account. Close the shell when you are done.

```
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.



✓ Enable version control interface for RStudio projects

Git executable: /usr/local/bin/git

SVN executable: /usr/bin/svn

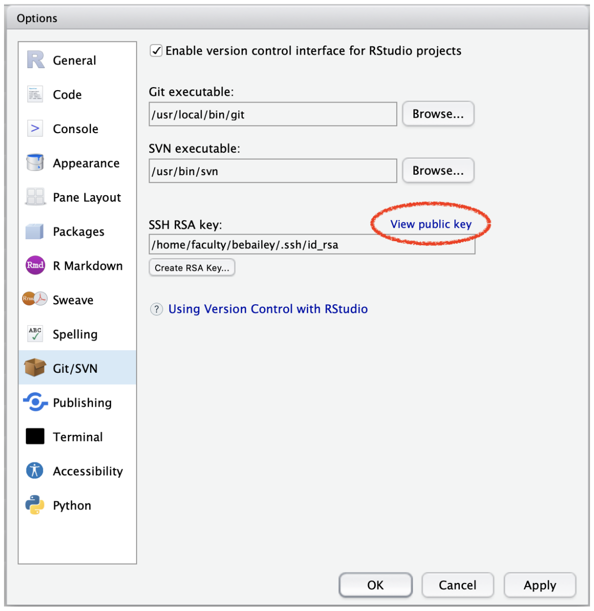
SSH RSA key: (None)

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).



✓ Enable version control interface for RStudio projects

Git executable: /usr/local/bin/git

SVN executable: /usr/bin/svn

SSH RSA key: /home/faculty/bebailey/.ssh/id_rsa

   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 **profbailey-course-content** repo on our class's GitHub page.

   b. Click on the **Code** button Code ▾, and select **Use SSH** (unless you've 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 "profbailey-course-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** (*this is what I do*): *Organize your github repos by course or major project.* One the RStudio server, for example, change the text from *"~"* (your home folder) to *"~/stat231"* by clicking "Browse" then creating a *stat231* 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 (once per device).

Workflow for individual assignments with Git and RStudio

This process will be the same whether you are on your own machine or the RStudio server.

Everyone in the class has access to the *profbailey-course-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 *profbailey-course-content* repo and your own personal repo for the course.

1. Use the **Files** pane in RStudio to navigate to the *profbailey-course-content* repo and open the repo project (*profbailey-course-content.Rproj*).

2. *Pull* any changes to the repo by clicking on the Git toolbar menu or the Git pane in RStudio and then clicking **Pull**. Remember, you don't have permission to push any changes to the *profbailey-course-content* repo!

   Once you've cloned the repository (following the directions above), you will want to pull in changes each time a new lab or homework is added. If you get an error about conflict, don't freak out! This can happen if you accidentally edit one of the files in the *profbailey-course-content* repo.

   You shouldn't be editing the files in this repo; rather, you should copy files to your own repo before making changes. That being said, I'll also try my best not to edit files too much after an initial material post.

3. *Copy* the relevant file to your own repo:

   a. In RStudio, on the **Files** pane, navigate to the *profbaily-course-contents* repo, and check the box next to the file you want to copy (usually within the "homeworks" or "labs" folder).

   b. Click on **More** → **Copy to...** and navigate to the appropriate folder (e.g., *homework*) in your personal repo where you want to save the file.

   c. Click save and then close the *profbailey-course-content* repo.

4. Still in RStudio, navigate to your own repo and open the corresponding repo project.

5. In the **Git** pane, you should see the new file that you added to your repo. Check the box next to the file, and then click the **Commit** button. Add an appropriate commit message (e.g. "Add hw0 file"), and click **Commit**. Then, click **Push**.

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

6. Open the file you want to work on and get started! After you make changes to an assignment, *commit* them.

   What are commits? Commits are essentially snapshots of your projects. Commits save this snapshot to your local version of Git (located on your hard drive or the Amherst server). For example, if I add code to a file so that it prints "Hello world" and then commit the file with an informative message, I can look at the history of my commits and view the code I wrote at that time. If I made some more changes to the code later that resulted in an error, I could go back to the commit where the code was originally working. This way you don't need to create several versions of your homework (homework-v1, homework-v2, . . .) nor do you have to try to remember what your code originally looked like.

   Four things about committing:

   (1) You should commit somewhat frequently. At minimum, if you're doing a homework assignment, you should make a commit each time you've finished a question.
   (2) You can make commits in RStudio using the Git toolbar (make sure the toolbar is visible by going to the *View* menu → *Show Toolbar*) or in the Git pane. Check the file(s) you want to commit and click *Commit* (either in the Git toolbar dropdown menu or in the Git pane in the top right window). The R Packages site provides more detail about how this works.
   (3) 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.
   (4) To get you started with good habits, one-line commit messages should start with a capital letter, hould not end with any punctuation, and should complete the sentence, "If applied, this commit will *commit message...*"

7. At some point you'll need to *push* the updated version of the assignment back onto GitHub. This way the instructor can help you with the latest version of your code, or you can work on your assignment on a different device (e.g., switch over to the RStudio server from your local machine), or your collaborators can see and pull your changes before making and pushing their own edits. You should push work frequently when you have a shared GitHub repo for project collaborations (i.e. more than one person is working on a project and code).

   When you are ready to push, you can click on the Git toolbar dropdown menu or the Git pane in RStudio, and then click **Push**. You can also do this after each commit in RStudio by clicking **Push** in the top right of the *Commit* pop-up window.

Closing advice

1. Pull often!

2. Do not add pdfs (or other files that are generated by your Rmd) to GitHub until you are ready to submit an assignment. When the time comes, consider giving the pdf a unique name. For example, if your file is *lab01.Rmd* which generates *lab01.pdf*, rename this file as *YourName_lab01_2021-09-01.pdf* before pushing.

3. Do not add large files (50mb or larger) to GitHub. Things will break.

4. If you get stuck or run into a conflict when pushing or pulling, try running `git status` at the shell (**Tools** → **Shell**).

5. Really stuck? Please contact me or the Statistics and Data Science Fellows (available Sunday through Thursday 7–9pm ET in SCCE E208).

Additional resources

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

GitHub Guides