Introduction to Git and Github Classroom

Jonathan GIlligan

August 19, 2017

Revision Control for Reproducible Research

A very important part of reproducible research is using revision control.

For an excellent introduction to using git with R, I recommend Professor Jenny Bryan's lecture, "Happy Git and Github for the useR." Her slides for the lecture are available at http://happygitwithr.com/, and you can watch the lecture online at https://www.rstudio.com/resources/videos/happy-git-and-gihub-for-the-user-tutorial/ (It's two hours long and walks you through using git with RStudio in detail and gives lots of practical advice)

Installing Git on Your own Computer

Git is installed on the lab computers, but since we will use git to submit lab reports, it will probably be a good idea for you to install git on your personal computer as well. There are three good options for git that work

- You can download Git from https://git-scm.com/. This is a pretty bare-bones distribution.
- There is a very popular free git client called "Source Tree" that has some very nice graphical utilities that let you use git from Windows Explorer or Mac Finder. You can get Source Tree from https://www.sourcetreeapp.com/ for Mac or Windows. You will need to sign up for a free account at Atlassian to install it.
- A third option is Git Kraken, which you can get from https://www.gitkraken.com/ Git Kraken is very popular, and also has nice graphical interface and integration with Windows Explorer and Mac Finder. It is free for educational, personal, and other non-commercial use.

After you install Git, it is important to run two commands:

- Open a git command line:
  - On Windows, open the program menu, go to "Git" and click on "Git Bash"
  - On a Mac, open a terminal window
- Run the two following commands:
  - `git config --global user.name "Your Name"` (using your own name instead of "Your Name")
  - `git config --global user.email "your.email.address@vanderbilt.edu"` (using your own email address)

Git uses information to keep track of who makes changes to a file. If you are editing a file on your computer and a friend is editing it on her computer, git uses this user information to keep track of who made each change. Then when you and your friend merge your changes, git will be able to tell you which of you edited what.

What Git Does

Git is a very powerful tool that can do many things, and can become very confusing, even to experts. Fortunately, we can ignore most of what git can do, and focus on a few simple things:

- Cloning a project from a remote server (e.g., github.com) to make a local copy of the file repository.
- Using file differencing to see what you changed since the last time you committed changes to your local repository
- Staging and committing changes that you made on your local computer to your local file repository
- Synchronizing Pushing changes between your local computer and a remote server.

Pretty much everything you might want to do with git, you can do from inside RStudio.

Git Vocabulary:

- **Repository** is where git stores the history of an entire project. Git tracks every change that you make to every file in a project.

  Git can synchronize repositories in different computers. We will use github a lot for the labs in this course. At the beginning of a lab session, you will clone the repository for the lab from github. Cloning makes a copy of the repository from the remote computer to your local computer. After you have cloned a repository, you will have not only the current version of all the files in the project, but you will have the entire history of each of those files.

  If you are collaborating with other people, you can both edit files and then synchronize your repository with your partner's repository and this will let each of you see all of the changes that each of you made to the files.

  An easy way to clone remote repositories from RStudio is to go to the "File" menu and choose "New Project". Then choose the option, "Version Control". Then select "Git" and enter the URL for the remote repository.

- **Commit** A commit is a snapshot of all the files in a repository at some point in time. If you edit some files, create some new files, and delete some files, then you can commit all of those changes (edits, new files, and deleted files). The git repository will add the new files to the repository, note that the deleted files have been deleted, and note the changes in the edited files between the latest version in the repository and the edited version that you are committing. Then it will then note the current state of all the files in the repository, so the commit represents a snapshot of the current state of all the files in the repository when you make the commit.

Using Git with RStudio

- **File Differencing** When you are working in an RStudio project that has a git repository, if you edit a file, RStudio notices that it has changed and the file appears in the "Git" window in RStudio. If you highlight that file in the "Git" window and click on the "Diff" button, RStudio will open a window where it will show you what changed in the file, compared to the latest version in the repository.

  RStudio shows the changes by identifying which lines in the file have changed, and showing the old version of those lines in red, and the new version in green. If you delete a line, you will just see a red line, and if you add a new line, you will just see a green line.

  If you decide that you are not happy with the changes and want to restore the file to the version that existed in the repository, you can right click on the file and select "Revert." Be careful with this, because **if you revert a file, you will lose *all* the changes you made!**

- **Staging and Committing Changes** is where you tell git to save the changes that you made to your local files into your local repository. Git will only record changes in files when you commit changes. Committing is a two-step process: first, you tell git which files you want to commit (that is, which files you want to record changes for), and then once you have selected the files, you tell git to commit the changes on those files to its memory in the repository.

In RStudio, any file you change will show up in the "Git" window. Changes can be editing a file, creating a new file, or deleting a file. You stage a file by checking the box next to the file name in the "Git" window in RStudio. Then you tell git to permanently remember those changes by Committing: Click on the "Commit" button in the "Git" window.

Committing stores changes. If you delete a file, and then stage and commit it, the repository will note that the file is now deleted. However, all of the previous versions of the file, before you committed the delete, will be stored in the repository. This is very useful. If you have ever been working on a project and accidentally deleted an important file, that can be very painful. With git, **if you committed that file to the repository, then even if you delete the file and commit the delete, you will be able to recover the file by checking a previous version out of the repository.**

Note that the repository **only remembers the changes that you tell it to commit.** Specifically, it records the differences between the last version of the file that you committed and the new version that you are committing. It does not know anything about anything that happened between the two commits. Thus, if you edit a file but do not commit it, and then delete the file and commit the file as a deleted file, git will only record the fact that you deleted the file. It will not remember any edits that you made before you deleted it.

It is a good idea to commit changes pretty frequently. Any time you hve something that is working, it's a good idea to commit. For instance, if you are working on a lab project that has many parts to it, as soon as you have answered one part you should stage and commit the changes. That way, if something goes wrong, you can recover your work from the repository.

When you commit changes to a repository, git asks you to enter a comment to describe the commit. You can give a brief description of what you changed in the commit, or remark on the state of the files (e.g., "Answered exercises 1-5." or "Finally, the scripts are working properly!"). Think about what would be useful to you in helping you understand the commit if you are looking back over your repository history at some time in the future.

- **Synchronizing Repositories** Git can synchronize multiple repositories. You can **push** the changes you have made on your local repository to a remote repository on a server, or you can **pull** changes in the remote repository to your local computer and merge them into your local repository.

  When you work on the computer in the computer classroom where our lab sessions meet, your edits are stored on the local repository on that computer. You turn in your lab work by pushing your local repository to the remote repository at github.com. Only after you have pushed your work will I be able to see what you have done.

  If you have a copy of the lab project repository on the computer in the computer classroom and another copy on your personal computer, you might make some edits on the classroom computer and some edits on your personal computer. Now you want to synchronize the edits you made on the two computers. You do this as follows:

  1. **Commit** any changes that you want to keep on the lab computer. You will not be able to proceed if there are any uncommitted changes, so you will need to either commit or revert any changed files.
  2. **Commit** any changes that you want to keep on your personal computer.
  3. **Pull** any changes from the remote github repository to the lab computer. If nothing has changed in the remote repository since the last time you synchronized the repository on the lab computer, nothing will happen. However, if anything changed in the remote repository, then this will merge the changes to the remote repository with the changes that you committed on your local computer.
  4. **Push** the changes from the lab computer to the remote github repository. This will make sure that the remote repository is identical to the repository on the lab computer.
  5. **Pull** the changes from the github repositiroy to your personal computer. This will merge the changes from the lab computer's repository (which you pushed to the github repository) with the changes that you have committed on your personal computer.

6. **Push** the changes from your personal computer to the remote repository. This will send the changes on your personal computer to the remote repository on github to make sure that the remote repository on github is now identical to the repository on your personal computer.
7. **Pull** the changes from the remote github repository to the lab computer. This merges the changes from your personal computer into the repository on the lab computer.

After the last step, the three repositories (personal computer, lab computer, and github) will all be identical, and will include all the changes that you made on your personal computer and the lab computer.

This may seem complicated, but you can simplify it if you follow a basic practice:

- Every time you start working on a project that has a remote repository, **pull** from the remote repository before you start working.
- Every time you have committed work that you don't want to lose, **push** to the remote repository.

  This is particularly important for the lab computers because if the computer you are working on crashes or reboots, or if you log out, you will lose all of your files on the lab computer. **If you push your commits to the remote repository on github, then your work is preserved.**

  This also means that **if you push projects from your personal computer to a remote repository (e.g., on github), then even if your personal computer breaks or gets stolen, the remote github repository will have the whole history of the project, up through the last time you pushed it.**

Conflicts

If you edit the same file on two different computers, git will attempt to merge the two sets of edits automatically. Git does a good job with this if you edit different lines on the two computers. However, if you edit the same lines on the two computers, git doesn't know which version of the changed lines you want to keep.

Original	Computer 1	Computer 2
Mary had a little lamb	Mary had a **great big lamb**	Mary had a little lamb
Its fleece was white as snow	Its fleece was white as **clouds**	Its fleece was white as **milk**
And everywhere that Mary went	And everywhere that Mary went	And everywhere that Mary **walked**
The lamb was sure to go	The lamb was sure to go	The lamb was sure to go

If you try to merge these, git can deal with the edits to the first and third lines, but the two computers made incompatible edits to the second line and git does not know whether to go with "clouds" or "milk".

When you pull the changes from one computer onto the other, git will complain about a conflict, and the file will look like

```
Mary had a great big lamb
<<<<<<< HEAD
Its fleece was white as clouds
=======
Its fleece was white as milk
>>>>>>> change
And everywhere that Mary walked
```

```
The lamb was sure to go
```

Then you have to manually edit the file to resolve the conflict. There are graphical tools to help you manage merge conflicts (this is one of the reasons people like to use graphical git tools like Source Tree or Git Kraken).

If you have conflicts, you will need to edit the files to resolve the conflicts and delete the lines git uses to mark conflicts (the ones beginning with `<<<<<<<`, `=======`, and `>>>>>>>`). Then you will need to stage the files where you resolved the changes and make a commit.

Github and Github Classroom

Github is a web site devoted to sharing open-source git repositories and allowing paying customers to operate private git repositories.
As a student, you can get a free educational account that allows you an unlimited number of public repositories for your projects and a limited number of private repositories.

Github classroom is an add-on service that github offers for teachers, which allows teachers to post assignments on github and then invite students to clone the assignment and then turn in the completed assignment via a private repository.

For each lab assignment, I will create a repository on Github Classroom and invite you to accept the assignment. When you accept the assignment, Github will clone the assignment into private repository just for you on github. Only you, I, and the teaching assistant will be able to see your private repository.

You can then clone the private repository to your personal computer or a computer in the laboratory classroom and complete it. As you make commits, I encourage you to push the changes back up to github. When you are done with the assignment, you will create a "pull request"