Introduction to Git and Github Classroom

Jonathan GIlligan

January 23, 2018

Revision Control for Reproducible Research

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

Installing Git on Your own Computer

To get started, I would like you to install git on your personal computer.

There are two good options for git that work

- Git Kraken, which you can get from https://www.gitkraken.com/, is a very popular tool, which has nice graphical interface and integration with Windows Explorer and Mac Finder. It is free for educational, personal, and other non-commercial use.

  This is what I recommend.

- If you don't want to use Git Kraken, you can install a bare-bones version of git that runs from the Windows command prompt, or from the Terminal on Mac or Linux systems, but this is much more fussy.
  - For Windows, you can download and install git from https://git-scm.com/download/win.
  - Many Macs have git installed already and if you are running Mavericks (10.9) or above you can test whether git is installed, and automatically install it if it isn't simply by trying to run git from the Terminal the very first time.

    ```
    $ git --version
    ```

    If you don't have it installed already, it will prompt you to install it. If necessary, you can manually install the XCode command line tools, which include git, by typing

    ```
    xcode-select --install
    ```

    in a Terminal window.

    If you want a more up to date version, you can also install it via a binary installer. A macOS Git installer is maintained and available for download at the Git website, at http://git-scm.com/download/mac.
  - For Linux distributions, git is easily installed from your distribution's package manager. On Ubuntu and Debian type distributions, open a terminal window and type

```
$ sudo apt-get install git-all
```

On Red Hat, Fedora, and CentOS type distributions, open a terminal window and type

```
sudo dnf install git-all
```

After you install Git, it is important to tell git who you are, so it will know how to identify you as the author of changes you make:

- If you are using Git Kraken, open the "preferences" (Open the `File` menu, choose `Preferences` or use the keyboard shortcut `Ctrl-Comma` or `Cmd-Comma` on a Mac: Ctrl or Cmd key and `,` (the comma)).

  In Preferences, click on `Profiles` on the left-hand window and then make sure the `Keep my git config updated with my profile info` box is checked and make sure your profile has your correct name and the email address you want to use.

- Alternately, if you are using the command-line version of git:
  - Open a git command window:
    - On Windows, open the start menu, go to "Git" and click on "Git Bash"
    - On a Mac or Linux, 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)
  - You may also want to add the following, which makes it easier to edit commit comments when you are committing from the command line:
    - on Windows: `git config --global core.editor nano.exe`
    - on a Mac or Linux: first type
      `which nano`
      which will tell you where the `nano` editor is located, and then type
      `git config --global core.editor /bin/nano`
      or
      `git config --global core.editor /usr/bin/nano`
      or substituting whatever the result of `which nano` is for `/bin/nano`

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.

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

- **File Differencing** When you have Git Kraken open in a project with a git repository, if you edit a file, Git Kraken notices that it has changed and the file appears in the "Unstaged Files" pane on the upper right of the Git Kraken window. If you highlight that file in the "Unstaged Files" pane Git Kraken will show the changes you have made in the main display pane.

  Git and Git Kraken show the changes by identifying which lines in the file have changed, compared to the version in the repository. It shows the old versions 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 in the "Unstaged Files" pane and select "Discard Changes". Be careful with this, because **if you discard changes for a file, you will lose *all* the changes you made since the last time you committed it to the local repository!**

- **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:

  1. First, you tell git which files you want to commit (that is, which files you want to record changes for). This is called "staging" the files.
  2. Then, once you have staged the files you want to keep, you tell git to commit the changes on all of those files to the repository, which makes a permanent record of the changes.

  – **Staging Files:** Staging is where you select which files you want to include as a commit to the repository.

  In Git Kraken, any file you change will show up in the "Unstaged Files" window. Changes can be editing a file, creating a new file, or deleting a file. You stage a file by right clicking on a file in the "Unstaged Files" window and selecting "Stage". You can also click on the file in the "Unstaged Files" pane and then click the "Stage File" button on the top right of the main file pane or the "Stage File" button on the right of the filename in the "Unstaged Files" pane. You can stage all of the changes on the project (multiple files) by clicking the "Stage all changes" button on the top right of the "Unstaged Files" pane.

  If you are using the command-line version of git, you would stage files by saying `git add <filename>`, where you would put the actual file name in place of `<filename>`. You can commit all of the files in the current folder and all of its subfolders with `git add .`.

  If you need to unstage a file from the command line, you can type `git reset <filename>`.

  – **Committing Files:**
  Committing permanently stores the changes to a set of one or more files in the repository. 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 retrieving a previous version of the file from the repository.** Similarly, if you edit a file and then decide that you don't like the changes you made, you will be able to retrieve any older version that you committed to 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 have staged. 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 project that has many parts to it, as soon as one part is working, you should stage and commit the changes. That way, if something goes wrong, you can recover your work from the repository in which the first part is working.

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., "Function foo is now working, but function bar still needs work." 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.

To commit files in Git Kraken, you would type a descriptive message or comment in the box marked "Commit Message" at the bottom right of the Git Kraken window (below "Staged Files"), and then you would click on the "Commit" button at the bottom right.

To commit files with the command-line git, you would type `git commit -m "commit message"` (where you would put a real message instead of "commit message"). If you leave off the `-m "commit message"`, git will open a text editor for you to type the commit message. By default, git uses an editor called "vi", which can be very confusing to use. This is why I recommended above that you tell git to use the "nano" editor instead, because nano is much easier to use.

- **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 your personal computer, your edits are stored on the local repository on that computer. You will turn in homework assignments 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 and a friend are both working on the project on your own computers, you will have a copy of your project repository on your computer and your friend will have another copy of the repository on her computer. You might make some edits on your computer and your friend might make some edits on her computer.
  Now you want to synchronize the edits on the two computers. You do this as follows:

  1. You should **Stage** and **Commit** any changes that you want to keep on your computer. You will not be able to proceed if there are any uncommitted changes, so you will need to either commit or discard changes for any changed files.

2. Your friend should **Stage** and **Commit** any changes that she wants to keep on her computer.
3. You should **Pull** any changes from the remote github repository to your computers. If nothing has changed in the remote repository since the last time you synchronized the repository on your computer, nothing will happen. However, if anything changed in the remote repository, then git will merge the changes from the remote computer with the local repository on your computer.
4. **Push** the changes from your computer to the remote github repository. This will make sure that the remote repository is identical to the repository on your computer.
5. Your friend should now **Pull** the changes from the github repositiroy to her computer. This will merge the changes from your computer's local repository (which you pushed to the github repository) with the changes that she committed to the local repository on her computer.
6. Your friend should now **Push** the changes from her computer to the remote repository. This will send the changes on her computer to the remote repository on github so that remote repository will now be identical to the repository on your friend's computer.
7. Now you should **Pull** the changes from the remote github repository to your own computer. This merges the changes from your friend's personal repository (which she pushed to github) into the repository on your computer.

After the last step, the three repositories (your computer, your friend's computer, and github) will all be identical, and will include all the changes that each of you made.

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 will also protect you from losing your work if your computer crashes or if your disk dies. **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 lost or stolen, the remote github repository will have the entire 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

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 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. You can get a free account at https://github.com and as a student, you can get some free extra features if you request a student account at https://education.github.com/discount_requests/new.

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 computational 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.