A Brief Introduction to GitHub for Social Scientists using Stata and Dropbox[*]

Heitor S. Pellegrina

May 16, 2018

1 Introduction

GitHub is a widely used online platform that computer scientists use to keep track of their codes and to collaborate on their projects. The word GitHub is the combination of Git with Hub. Git is a version control system that stores all the history of a code. With Git, programmers can check what they changed in their codes and when they changed their codes. They can also restore previous versions of their projects if they do not like their new versions. GitHub is a web-based repository that keeps all the history of a code on a cloud using the Git system. Since everything is in a cloud, it allows many researchers to colaborate on a specific project.

1.1 Why should I use GitHub+Dropbox instead of just using Dropbox?

Dropbox is a great resource for teamwork. It allows many people to work on the same project and everything is automatically synchronized across computers. However, as projects expand in terms of size and collaborators, one may run into problems. First, if you have used Dropbox to work with many collaborators you probably noticed the generation of "conflict" copies. This happens when Dropbox is not sure which version it should use because there is more than one synchronization happening at the same time. As I explain below, GitHub tends to avoid this problem because files are not automatically synchronized. Second, Dropbox lacks a more sophisticated system to track what changes in each version of a code. If something changes, we do not know who changed a

[*]The purpose of this material is to serve just as an introduction to Github. I hope that this material can be useful for Social Scientists who are trying to integrate Github with Dropbox for the first time. This document should not be considered as representative of high standards of project management! For an advanced template of project management using GitHub, see the material here click here. I thank Yaw Nyarko, Andres Ramirez and Martin Smith for comments. Contact: heitor.pellegrina@nyu.edu

code and what exactly they changed in it. GitHub has an interface that makes it easy to identify these changes. Third, if you want to bring back previous versions of a project, GitHub can do it quickly, but with Dropbox, we can only bring back specific files one at a time. While some of these problems may be more likely to appear in large projects (imagine writing the codes for facebook for example...), as big empirical projects become more frequent in Economics, I believe that the use of Github will inevitably become more frequent as well. The main cost of using GitHub is that you have to be much more systematic about the way how you work since you have to manually synchronize your changes with the cloud.

1.2 What is the difference between Dropbox and GitHub?

For me, the main difference between Dropbox and GitHub is that now we have an intermediary step for the synchronization of files called *"commit"*. In Dropbox, we have

$$\text{Master Repo (on the web)} \Longleftrightarrow \text{Local Repo (in your computer)},$$

where Master Repository is the cloud where the codes are and local repository is a copy of the Master Repository that you keep in your computer. If someone changes something in their Local Repository, things are automatically synchronized with the Master Repository and across all the Local Repositories of other researchers working on the project. Therefore, if someone makes a mistake, this mistake spreads across the project and there is no straighforward way of identifying this mistake and going back to previous versions of the project. Now, with Git, we have an intermediary step

$$\text{Master Repo (on the web)} \Longleftrightarrow \text{Commit} \Longleftrightarrow \text{Local Repo (in your computer)},$$

The way how synchronization works with Github is as follows. You first download all the repository that cointains the codes for the project to your computer. If it is the first time that you are downloading a project, you use a process called *"clone"*. If you are just updating the project to your local repo, you use a process called *"Pull"*. Both are essentially downloading the files from the cloud. After you make your changes to the files, you can then upload them to this intermediary step called commit using a process called *"add"*. Once you added everything that you want, you use a process called "commit", which basically packages all the "*adds*" that you made. You can upload this commit to the cloud using a process called "*Push*". GitHub keeps track of all the versions of the project. If someone makes a mistake, we can easily go back to the version of the project that existed before the mistake.

1.3 Can I use GitHub as a repository for my datasets?

GitHub is a tool for keeping track of codes, but it is not supposed to be a repository for datasets. This is where an integration of Dropbox with Github is useful. GitHub can keep track of the codes that are written for a project, and we can leave the datasets and output of the project (figures, tables and pdfs) untracked. For example, a researcher can keep track of do-files that clean the data, but only keep the final version of the cleaned dataset. The idea is that, by keeping track of the code, we can always go back to previous versions of the project and re-obtain previous versions of the cleaned dataset.

1.4 Prerequisites for this manual

Before you move forward, make sure that you do the following

- Go to GitHub and set up an account (click here).
- Follow their introduction guide.
- Go to Git website and install git (click here).

2 Synchronization using GitHub

There are two ways to synchronize your files with the master repository. First, you can use a software called GitHub Desktop. This is a software where you can "point-and-click" your way through synchronization. I found this software quite user friendly. Second, you can use the Terminal in Mac[1] (or the command line in windows). They may seem scary at first, but, in practice, you do not have to be a sophisticated programmer to use it. There are just a few commands that you have to use for your daily work. I see two main benefits in learning how to actually type the commands in Terminal. First, whenever you want to do something specific and you google about it, the answers that you get are mostly based on command lines. Therefore, to understand the answers that are given by the online community, you need to have a basic understanding of the language that they use. Second, there are some functionalities that I was not able to obtain with the graphical interface.

2.1 Basic commands (MAC)

- *ls* (this command will **list** the files in the current directory)

[1]If you are using a Mac computer and do not know what Terminal is, read this website here.

- *cd Dropbox* (this command will change the **current directory** and enter the folder called "Dropbox")
- *cd ..* (this will go a folder back in the hierarchy of folders)
- *mkdir projects-git* (this command will **make a directory** called projects-git)
- *git clone https://github.com/hpellegrina/hello-world* (this command will make a clone of the online repository called hello-world to the current directory in your terminal)
- *git add table_1_summary_stat.do* (this command will add the do.file called *table_1_summary_stat.do* to the commit that you are preparing)
- *git remove table_1_summary_stat.do* (this command will remove *table_1_summary_stat.do* from the commit)
- *git add -A -m "Adding all files"* (this command **adds** to the commit all the new files that you have and creates a **message** associated with the commit saying "Adding all files")[2]
- *git commit -m "Adding table_1_summary_stat.do"* (this command will commit the changes that you made to the folder, the phrase inside the quotation marks is the comment that you are going to attach to your commit)
- *git push* (this command will **push** the commit that you prepared that is still in your computer to the git cloud)
- *git pull* (this command will **pull** the current version of the project to your desktop)
- *git status* (will show you the modifications/deletions that you made in your local repo)

2.2 How a typical day of work looks like?

- You start by going to the directory on the desktop where the project is located using the terminal. You then type *git pull* to update the directory with everything that everybody else may have changed.
- You make all the changes in the directory related to your task.
- You type *git add -A* to add all the changes that you made to your branch.
- You commit the changes using *git commit - m "Issue 7: added table_1_summary_stat.do"*

[2]When you are looking throught the commits of a project, you can see the list of commits and the messages associated with each of them

- You type *git push*.

3 Hello-World! I'm an Economist!

We will be following the tradition in computer science. Below you will have your first commit using GitHub, Stata and Dropbox. I taylored this commit to include some of the tricks to integrate Dropbox with GitHub.

1. Before you start, make sure that you have a GitHub account and that you have installed Git into your machine. To double check if it is properly installed, go to terminal and type *git*. If it is already installed, it will show a list of commands that you can use.

2. Create a repository called "my-hello-world-econ" in github.com.

3. Create a folder in your Dropbox called "my-hello-world-econ-data-repo". This is the folder where the data will be stored.

4. Open the terminal. Connect your computer to your online account by typing in your terminal *git config --global user.email "email@example.com"*.

5. Go to your Dropbox folder in the terminal using the command *cd.*

6. Type *git clone https://github.com/hpellegrina/hello-world-econ* . This makes a copy of my repository into your computer.

7. Type *git clone https://github.com/yourusername/my-hello-world-econ.* This makes a copy of the repository that you created in Step 2 into your computer.

8. In the folder that you copied from me in Step 6, there is a folder called "datastore-for-practice". In this folder, there is a file called "macro_indicators.dta". Copy that file into "my-hello-world-data-repo" which you created in Step 3. You may do this using finder.[3]

9. Copy all the remaining files from "hello-world-econ" into "my-hello-world-econ".

10. In terminal, go to the folder "my-hello-world-econ". Delete the file called datastore. Type *ln -s /Users/UserName/Dropbox/my-hello-world-econ-data-repo datastor*e.[4]

[3]In a real project, we will avoid keeping any data file in "git" folders. I'm keeping this .dta file here in my git folder only for demonstration purposes.

[4]For PC users, you have to type *mklink /J datastore "C:\Users\ USERNAME\ Dropbox\my-hello-world-econ-data-repo"* in the Windows Command Prompter.

- This step creates a folder within “my-hello-world-econ” called “datastore” that is actually using the information from the folder “my-hello-world-econ-data-repo”. This is what is called a **symbolic link**.[5] By doing so, when you commit the files later you are going to push to the cloud a “fake” folder that does not have any data in it. The benefit of having this “fake” folder here is that when we write our codes it becomes easier to work with relative paths for a project.[6]

11. Run the do-file from your folder and create the new figure. You may have to change the location of the directory inside the .do file. Save the changes
12. Go back to terminal and type *git add -A*. With this command, you added your changes to the commit.
13. Type *git status*, check if everything that you changed is actually there for you to commit.
14. Type *git commit -m “Added first figure for Hello-World! I’m an Economist!”*.
15. Type *git push* to upload the commit to the master repo.
16. It is done! You can check your repository on GitHub.com in your account. It should contain all the files from this exercise.

4 Additional Features for Teamwork

4.1 Adding collaborators

To add collaborators into your GitHub repository, you can just use the github.com website.

- Click on the repository of the project.
- Click on the settings of the repository (one of the tabs on the top).
- Click on collaborators.
- Type the email of you collaborator and add him/her there.

[5]The definition given on the internet is “A symbolic link, also termed a soft link, is a special kind of file that points to another file, much like a shortcut in Windows or a Macintosh alias. Unlike a hard link, a symbolic link does not contain the data in the target file. It simply points to another entry somewhere in the file system.“

[6]Relative paths is when we write all links in our codes in relative terms. By doing so, we do not have to keep track of where exactly the files are located in your desktop. For example, consider that you are using Stata and you want to use a certain dataset that is located in “/Username/Dropbox/spatial-economy-project/datasets/mydata.dta”. This would be a problem if you are running a code from a computer with a different path to the data. You would have to change this line of command according to your computer. There is a way of organizing the project so that in your code you just have “../../datasets/mydata.dta”. Stata recognizes the directory of the code and finds the file “mydata.dta” **relative** to the directory where your code is located.

4.2 Issues

A useful tool for the organization of tasks in GitHub is the use of "Issues". Any collaborator in a project can create a new issue on the website associated with a repository and link their commits with these issues. For example, one can go to a repository and create an issue such as "Add clustered standard errors in table 1". When the issue is created it comes with a number. Let's say this number is 1. When someone in the project commits something related to the issue, this person can directly link his commit with the issue by writing in the beginning of the message #1. When you do so, the website will display a small notification under the issue with the user who updated the issue and also what was added there. You can also assign issues to specific people in the project.

4.3 Branches

Another useful tool for teamwork is the creation of branches. Let's say that you want to test the results from your paper using a restricted sample, but you do not want to affect your current results because it is possible that the restricted sample contains problems that you may find while working on it. In this case, you can create a "*branch*". This is a parallel repo on the cloud that your co-authors can use. In that case, we have

$$\begin{tikzcd}
\text{Master Repo (on the web)} \arrow[r, Leftrightarrow] \arrow[d, shift left] & \text{Commit} \arrow[r, Leftrightarrow] & \text{Local Repo (in your computer)} \\
\text{Branch Repo (on the web)} \arrow[r, Leftrightarrow] \arrow[u, shift left] & \text{Commit} \arrow[r, Leftrightarrow] & \text{Local Repo (in your computer)}
\end{tikzcd}$$

If the new sample works well, you may merge the branch with the master repo. If not, you can just delete the branch and the master repo will still be intact.