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

Heitor S. Pellegrina Andres F. Fajardo

June 5, 2018

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 and Hub. Git is a version control system that stores all the previous versions of files within a project. With Git, programmers can check what has changed in their codes and when those changes took place. They also have the ability to restore previous versions of their projects when they see fit. GitHub is a web-based repository that uses the Git system to save the complete history of a project in the cloud. Since everything is in the cloud, researches can collaborate on specific projects by sharing the changes they make on their local machines via the online repository.

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 creation 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 we explain below, GitHub tends to avoid this problem because files are not automatically synchronized. Second, Dropbox does not have a complete system to

*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. We thank Yaw Nyarko and Martin Smit for comments. Contact: heitor.pellegrina@nyu.edu or andres.fajardo@nyu.edu.

track the specific changes that are made across versions of a code. If something changes, we do not know who changed a code and what exactly they changed in it. GitHub has an interface that makes it easy to identify these changes. Third, if we want to restore 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 large empirical projects become more frequent in Economics, we 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 with your work, since you have to manually synchronize your changes with the cloud.

What is the difference between Dropbox and GitHub?

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

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

where Master Repository is the main copy of a project that is stored in the cloud and Local Repository is a copy of the Master Repository that you keep in your computer. Changes made by a collaborator on their local repository are automatically synchronized with the Master Repository and, in turn, with all collaborators' local repositories. Therefore, a single mistake by one collaborator can spread across all local repositories, and Dropbox lacks the capabilities to identify when exactly things went astray or to restore to a past working version of the project with ease. With Git, we have an intermediary step called commit:

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

The way how synchronization works with Github is as follows. You first download the Master repository to your local machine. 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 processes will download the up-to-date version of the project to your computer. After you make changes to the files, you can then load them to this intermediary step called commit using a process called *"stage"* or *"add"*. Once you added everything, you use a process called "*commit*", which basically packages all the "*adds*" that you made into a bundle of "approved changes". You can upload this commit to the cloud using a process called "*push*". GitHub keeps track of all the versions of the project for every commit. If someone makes a mistake, we can easily go back to the version of the project that existed before the commit that uploaded the

mistake to the Master Repository.

Can I use GitHub as a repository for my datasets?

GitHub is a tool for keeping track of codes, and it is not supposed to be a repository for datasets. This is why integrating Dropbox with Github is useful. GitHub can keep track of the codes of a project and light files (figures, tables and pdfs), and collaborators can share large data files using Dropbox.

How do we integrate Dropbox with GitHub in a local machine?

We integrate Dropbox with GitHub in a local machine as follows. For each project, we create a main folder that is tracked by GitHub, and which is a copy of the Master Repository online. Within this tracked folder, we create subfolders that are connected to external folders in the local machine that are synchronized with Dropbox. We establish this connection by creating symbolic links, which are basically shortcuts to a folder.[1] Given that the symbolic links target folders that are external to the main folder, GitHub will not keep track of the data files. See the picture in Appendix A for a illustration of this organization. Using this system, when we write our codes in Stata and any other program, we can reference to the "shortcut" folder to open external datasets.

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).

For Windows Users - Installing Git and using Git BASH

After downloading the installation file for Git, execute the installer and select the options as they appear in Appendix B.

If you use Windows, you can use a tool called Git BASH to work with the same commands that you would use if you had a Mac.[2] When this manual asks you to use 'the terminal', you should use Git BASH if you have Windows. There will be warnings when different commands are needed if working on a Windows machine.

[1] See this link for a helpful review of symbolic links.

[2] You may see references to *NIX on the internet. This is a reference to operating systems that are based on UNIX.

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. Second, you can use the Terminal in Mac (or the Command Prompt in Windows, see below). The Terminal 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 use for your daily work. We see two benefits in learning how to type the commands in Terminal. First, whenever you have a question about how to use Git, the answers posted online by other users will most likely involve using the Terminal. Second, there are some functionalities that we were not able to obtain with the graphical interface.

Basic commands (MAC)

- *ls* (this command will **list** the files in the current directory)
- *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")[3]
- *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)

[3]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

- *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)

How does a typical day of work looks like?

- You start by going to your project's local folder in a terminal window and download (pull) any changes your collaborators may have uploaded to the online repository.

```
$ cd local/path/to/the/project
$ git pull
```

- You work on a task, i.e. you make changes to the project's files.
- You check the changes you have made and select (add) the changes you want to approve into a bundle of changed files.

```
$ git status
$ git add -A
```

- You approve (commit) the changes and add a comment so that your collaborators know what the changes consist in.

```
$ git commit -m "Issue #1: added table_1_summary_stat.do"
```

- You work on other tasks and make additional commits (steps 3 and 4 above).
- You upload all the commits to the online repository.

```
$ git push
```

Hello-World! I'm an Economist!

We will be following the tradition in computer science. Below you will have your first commits using GitHub, Stata and Dropbox.

Exercise #0 - Getting started.

In this exercise you will set things up so that you have access to the files needed for the subsequent exercises.

1. Before you start, make sure that you have a GitHub account and that you have installed Git into your machine. To double check that Git is installed, go to terminal and type the following command. If it is already installed, it will show a list of Git commands.

```
$ git
```

2. Link git to your GitHub account.

```
$ git config --global user.email your@email.address
```

3. Make a copy of **hello-world-econ** into your computer. This will bring the files needed for this exercise.

```
$ cd /sample/path/for/files
$ git clone https://github.com/hpellegrina/hello-world-econ
```

4. There is a folder called **my-hello-world-econ-shared** in **hello-world-econ.** Copy this folder into your Dropbox folder. This is the folder you will use to share data, documentation, etc., with your collaborators.

Exercise #1 - Using GitHub to track codes.

In this exercise, you will use GitHub to track the codes and graphs of a new project: **my-hello-world-econ**. You will (1) create a remote repository of your project in GitHub, (2) configure git so that that it ignores *.dta Stata files, (3) set up a local repository to work on your project, and (4) upload the local changes to the online repository.

1. Go to your account in github.com. Create a repository called **my-hello-world-econ** and click on "initialize this repository with a README".

2. In the website of your repository, click on 'Create new file' to create a file called **.gitignore** in the main folder of the project**.** Input the content below in this file and click on 'Commit changes to the master branch'.[4]

```
1. # Ignore all *.dta files in the project
2. */.dta
```

3. Open a terminal window and use the following commands to create a local copy of the repository created in the step above. This will create a folder called **my-hello-world-econ** in your computer.

[4]Changes made to **.gitignore** will only apply to untracked files. If there are any files within these folders that git is already tracking, you will need to untrack them first. This can be the case if, for example, you already added a file to a commit. Here is an example of how to untrack files.

```
$ cd /sample/path/for/your/project
$ git clone https://github.com/yourusername/my-hello-world-econ
```

4. There is a **my-hello-world-econ-contents.zip** file in **hello-world-econ** (the repository you copied in Exercise #0). Extract the contents of this folder into **my-hello-world-econ**. Your local project folder should now have two subfolders: **analysis** and **data**.

5. The files you just added to **my-hello-world-econ** are local changes you made to the project. We will now update your online repository so that it incorporates these changes. First, let's add the changes to the "commit" stage (the second command below checks if the changes were properly added).

```
$ git add -A
$ git status
```

6. Create a commit and add a comment that explains what you are uploading.

```
$ git commit -m "Copied the contents of my-hello-world-econ-contents"
```

7. Push the commit to the online repository.

```
$ git push
```

8. Go to github.com and check that the changes are now displayed in your online repository.

Exercise # 2 - Using Dropbox to access data inputs.

Now, we are going to set up the folders that are connected to Dropbox using symbolic links as explained in section 1.4. We will (1) clean a dataset, (2) generate a figure and (3) upload your changes to the online repository.

1. As always, you must start by updating your local repository to download (*pull*) any changes (*commits*) made to the online repository by your collaborators.[5]

```
$ cd your/path/to/my-hello-world-econ
$ git pull
```

2. You will now create a folder for the data input that will be connected to an external folder. Go to the folder **data** in your project's data folder and create a symbolic link called **input** using the following commands.

```
$ cd your/path/to/my-hello-world-econ/data/
$ ln -s your/path/to/Dropbox/my-hello-world-econ-shared/data/input input
```

[5] See this for a discussion on pull vs fetch and rebase.

If you are using Windows, open the Command Prompt as an administrator (not Git BASH) and use the following commands. Note that we are using backslashes now and not forward slashes.

```
>> cd your\path\to\my-hello-world-econ\data\
>> mklink /D input your\path\to\Dropbox\my-hello-world-econ-shared\data\input
```

3. Open the file **make_urbanpop_dta.do**. Edit the path in the third line so that it points to its folder, that is, to **my-hello-world-econ/data/code**. Save and run the do-file. This step creates a cleaned dataset called **finaldata.dta**.

4. Open the file **fig_va_urbanization.do**. Edit the path in the third line so that it points to its folder. Save and run the do-file. This step creates a figures called **fig_hello_world.png**.

5. You will now upload (*push*) the changes you have made to your project to the online repository.

```
$ git add -A
$ git commit -m "Done with my first analysis!"
$ git push
```

6. Go to github.com and check that the changes in the online repository do not include the creation of *.dta files.

You have made your first contribution to my-hello-world-econ! Note that all the files in the external folder are not tracked by GitHub. Also, because we configured git to ignore *.dta files, **finaldata.dta** is not uploaded to the online repository.

(EXTRA) Using Python to create symbolic links.

In Exercise #2 you created symbolic links to connect your project to the the data inputs shared by collaborators on Dropbox. In projects with several datasets, it may be tedious to create a large number of symbolic links in the terminal when cloning a repository for the first time. We provide a Python script that checks all the data sources inside a major data folder and create symbolic links to each of them. This code works with the organization of folders presented in Appendix A. To run this code, open the terminal and type the commands below.

```
$ cd your/path/to/my-hello-world-econ
$ python3 input_slinks.py "your/path/to/project" "your/path/to/project-data"
```

Additional Features for Teamwork

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.

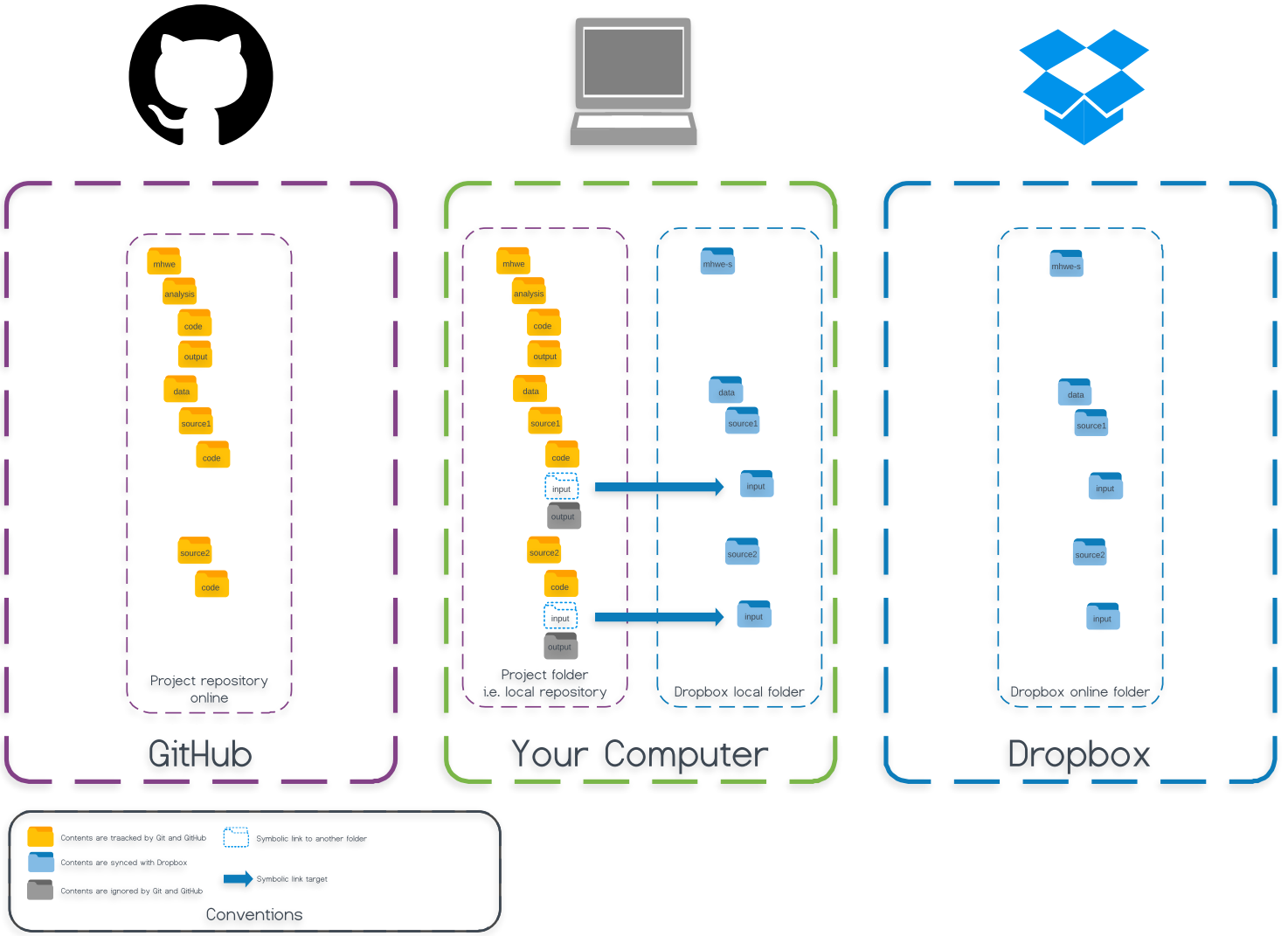
Branches

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

$$\begin{array}{ccccc} \text{Master Repo (on the web)} & \Longleftrightarrow & \text{Commit} & \Longleftrightarrow & \text{Local Repo (in your computer)} \\ & & \uparrow\downarrow & & \\ \text{Branch Repo (on the web)} & \Longleftrightarrow & \text{Commit} & \Longleftrightarrow & \text{Local Repo (in your computer)} \end{array}$$

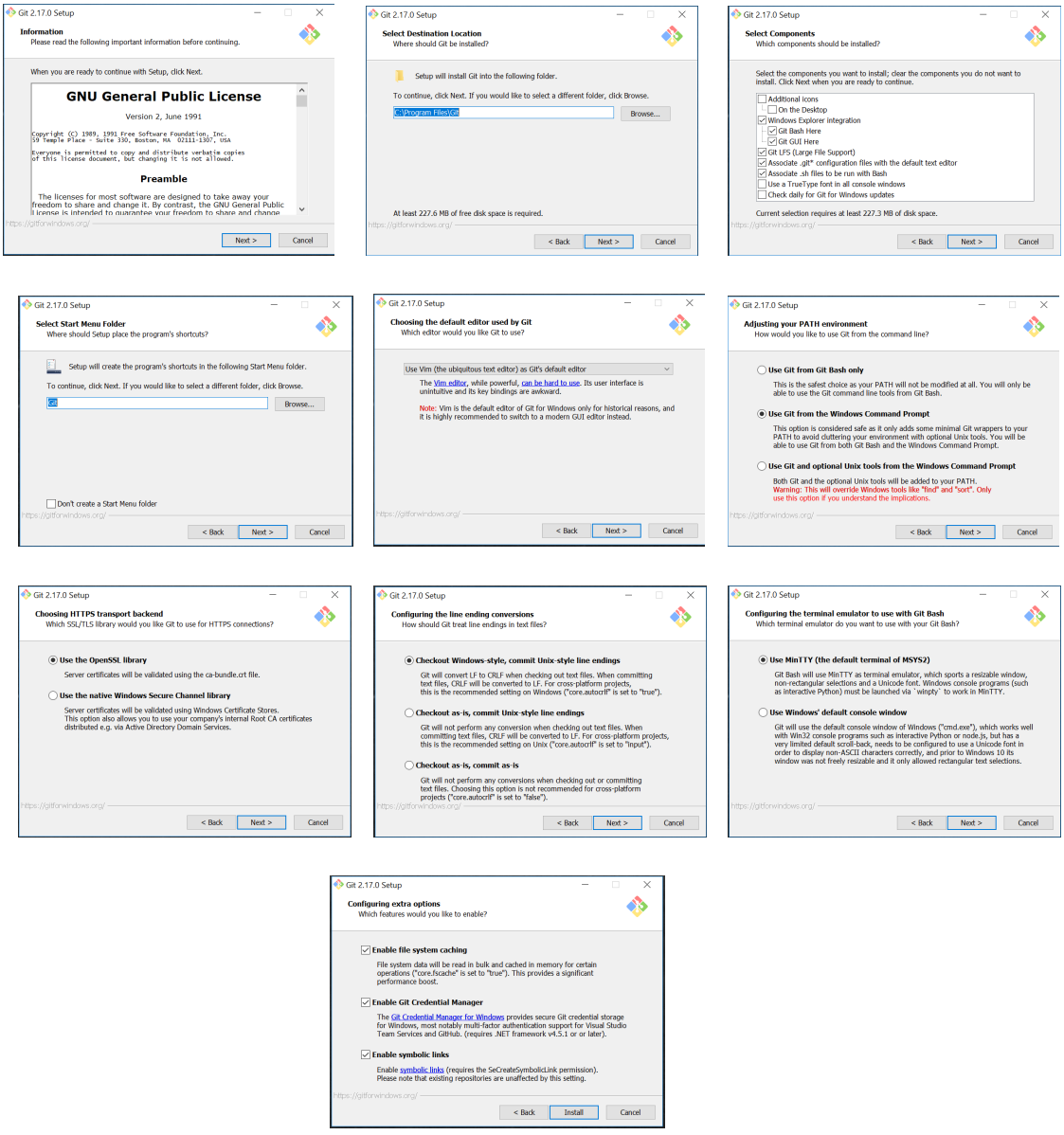
If the new sample works well, you may merge the branch with the master repo. If not, you can just delete the branch and the master repo will still be intact.

Big Picture


mhwe
analysis
code
output
data
source1
code
source2
code
Project repository
online
GitHub
mhwe
analysis
code
output
data
source1
code
input
output
source2
code
input
output
Project folder
i.e. local repository
mhwe-s
data
source1
input
source2
input
Dropbox local folder
Your Computer
mhwe-s
data
source1
input
source2
input
Dropbox online folder
Dropbox
Contents are traacked by Git and GitHub
Contents are synced with Dropbox
Contents are ignored by Git and GitHub
Symbolic link to another folder
Symbolic link target
Conventions

Windows installation screenshots

Figure 1: Windows installation options



Useful links

Here is a list of helpful online tutorials:

- Git & GitHub Crash Course For Beginners
- Git Basics: Merge and Rebase
- Git Handout - Houtan Bastani