

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

Heitor S. Pellegrina

Andres Fajardo

June 4, 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 collaborate 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 we explain below, GitHub tends to avoid this problem because files are not automatically synchronized. Second, Dropbox does not have a complete system to track what changes in each version of a code. If something changes, we do not know who changed a code

*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 and Martin Smith for comments. Contact: heitor.pellegrina@nyu.edu or andres.fajardo@nyu.edu.

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 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?

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

Master Repo (on the web) \Longleftrightarrow 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 straightforward way of identifying this mistake and restoring previous versions of the project. Now, with Git, we have an intermediary step

Master Repo (on the web) \Longleftrightarrow Commit \Longleftrightarrow Local Repo (in your computer),

The way how synchronization works with Github is as follows. You first download all the repository that contains 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 “*stage*” or “*add*”. Once you added everything, 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 for every commit. 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 of a project and light files (figures, tables and pdfs), and we can leave the datasets 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. 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 How we integrate Dropbox with GitHub?

We integrate Dropbox with GitHub as follows. For each project, we create a main folder that is tracked by GitHub. Within this tracked folder, we create subfolders that are connected to external folders in Dropbox where datasets reside. We establish this connection by creating “symbolic links”, which is basically a shortcut for a folder.¹ As such, GitHub keeps track of the main folder and all the “shortcut” folders, but since datasets are in an external folder, it does not keep track of the datasets. 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.

1.5 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](#)).

1.5.1 For Windows Users - Installing Git and using Git BASH

After downloading the installation file for Git, execute the installer and leave the default options in every step **except the last one**. At this point, check the option “Enable symbolic links.” Screen shots of this manual’s recommended installing options are available below in Appendix B.

After installing Git, if you use Windows, you can use a software 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

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

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

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.

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. Second, you can use the Terminal in Mac (or the command line 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 a specific task and you “google” your question, 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 we were not able to obtain with the graphical interface.

2.1 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

³When you are looking through the commits of a project, you can see the list of commits and the messages associated with each of them

- *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 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 further commits for each (steps 3 and 4 above).
- You upload all the commits you have made to the online repository.

```
$ git push
```

3 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.

3.1 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.

3.2 Exercise #1 - Using GitHub to track codes.

Remember the big picture: you are interested in GitHub because you want to track the changes made to your project's codes and light files such as output tables and graphs. In this exercise, you will use GitHub to track the codes and graphs of a new project: **my-hello-world-econ**. Specifically, 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) learn how to upload local changes to the online repository.

1. Go to your account in github.com and create a repository called **my-hello-world-econ**.
2. Click on 'Create new file' to create a file called **.gitignore** with the content below. Note that (1) this file should be in the main folder of your project, (2) the dot at the beginning of the name should be included, and (3) lines that start with '#' are comments. After creating the file, click on 'Commit changes to the master branch'.⁴

⁴Changes 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.

```
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 your repository created in the step above. This will create a folder called **my-hello-world-econ** in your computer.

```
$ 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. You will now update your online repository so that it incorporates these changes. To do so, you need to select (*stage* or *add*) the changed files you wish to update into a bundle of changes that you will then approve (*commit*) and upload (*push*) to the online repository. Let's start by staging the files to the commit. The second command below checks if the changes were properly added to the commit.

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

6. Create a commit and add a comment that explains the changes are being made.

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

7. Push the commit to the online repository. You may be asked to input your GitHub credentials.

```
$ git push
```

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

3.3 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. In this exercise, you will work with your first data input: **macro_indicators.csv**, which is inside **my-hello-world-econ-shared**.

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

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

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

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

3. Repeat the last step for the **agriculture** folder.
4. Open the file **import_population.do** and edit the path in the third line so that it points to its folder, that is, to **my-hello-world-econ/data/population/code**. Save and run the do-file.

```
1. clear all
2. * Write down the directory below
3. cd "your/path/to/my-hello-world-econ/data/population/code"
```

5. Repeat the last step for the files **import_agricultural_output.do**, **make_data_for_analysis.do** and **fig_va_urbanization.do** (in that order). You can find the last one in the **analysis** folder.
6. The codes you edited and ran produced new files. You will now upload (*push*) the changes you have made to your project to the online repository. Remember that we configured git to ignore *.dta files, so the files **population.dta**, **agricultural_output.dta** and **data_for_analysis.dta** should not be uploaded to the online repository.

```
$ git add -A
$ git status
$ git commit -m "Changed do-files"
$ git push
```

7. 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! Make sure you understand what each of the do-files are doing and how their relative paths work. The number of folders may seem excessive at first, but as a project grows you will find it helpful.

3.4 (EXTRA) Exercise # 3 - A shorter way to create multiple symbolic links.

In Exercise #2 you created symbolic links to connect your project to the the data inputs shared by collaborators on Dropbox. As a project gets bigger, however, it may be tedious to create a large number of symbolic links in the terminal when cloning a repository for the first time. In this exercise, you will run a Python script that creates symbolic links for each of your project's data folders to connect them to their Dropbox counterparts. The Python script does not create a symbolic link for the **finaldata** folder.

1. Go to your project's data folder and delete all the symbolic links you created in Exercise #2.
2. Open a terminal window, go to your project and run the Python script `input_slinks.py` with the arguments below.

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

3. Review the files and upload the changes to the online repository (see Step 6 in Exercise #2).

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.

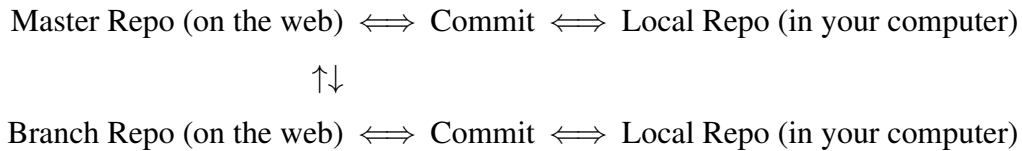
4.2 Issues

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.

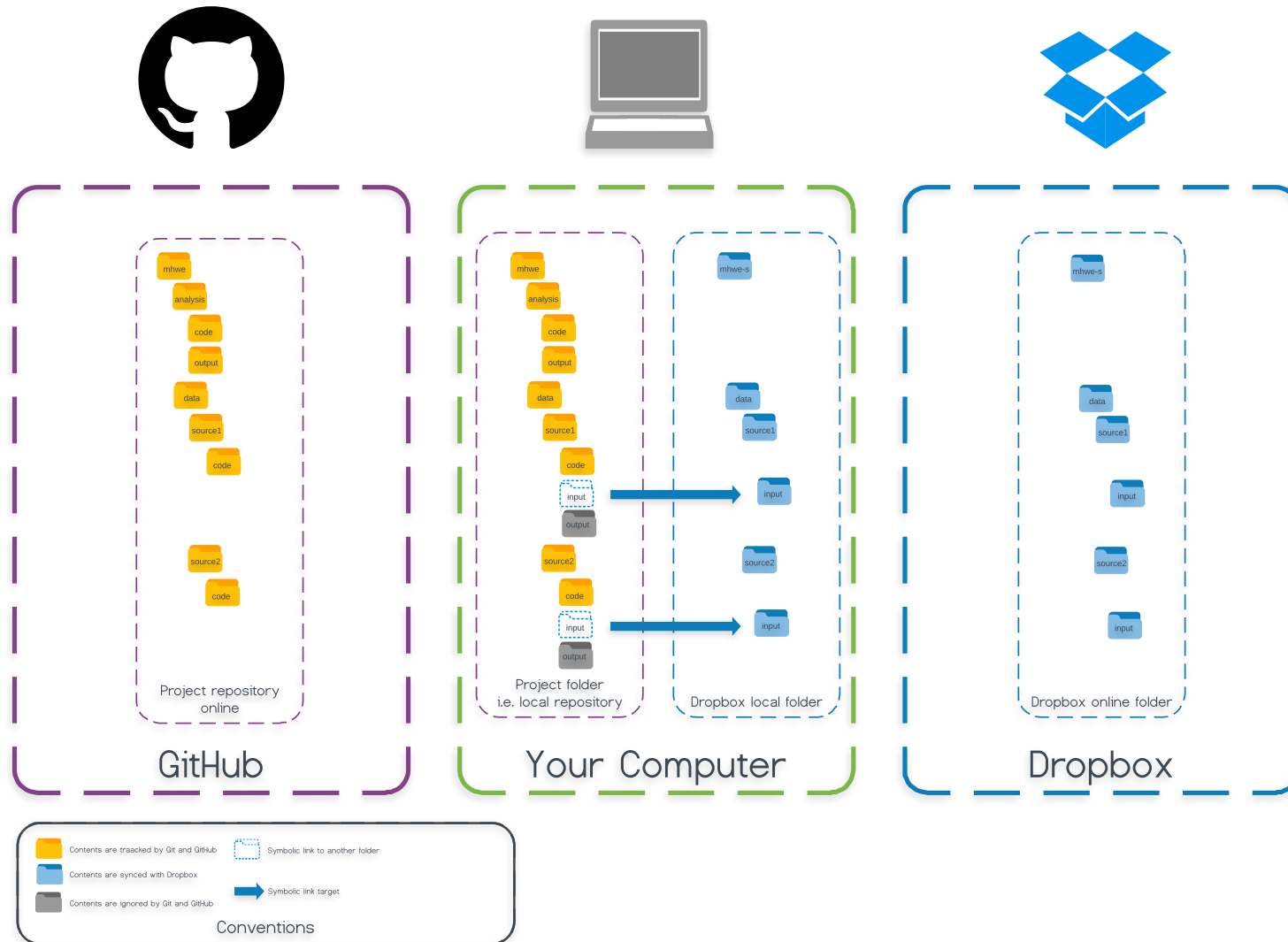
4.3 Branches

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



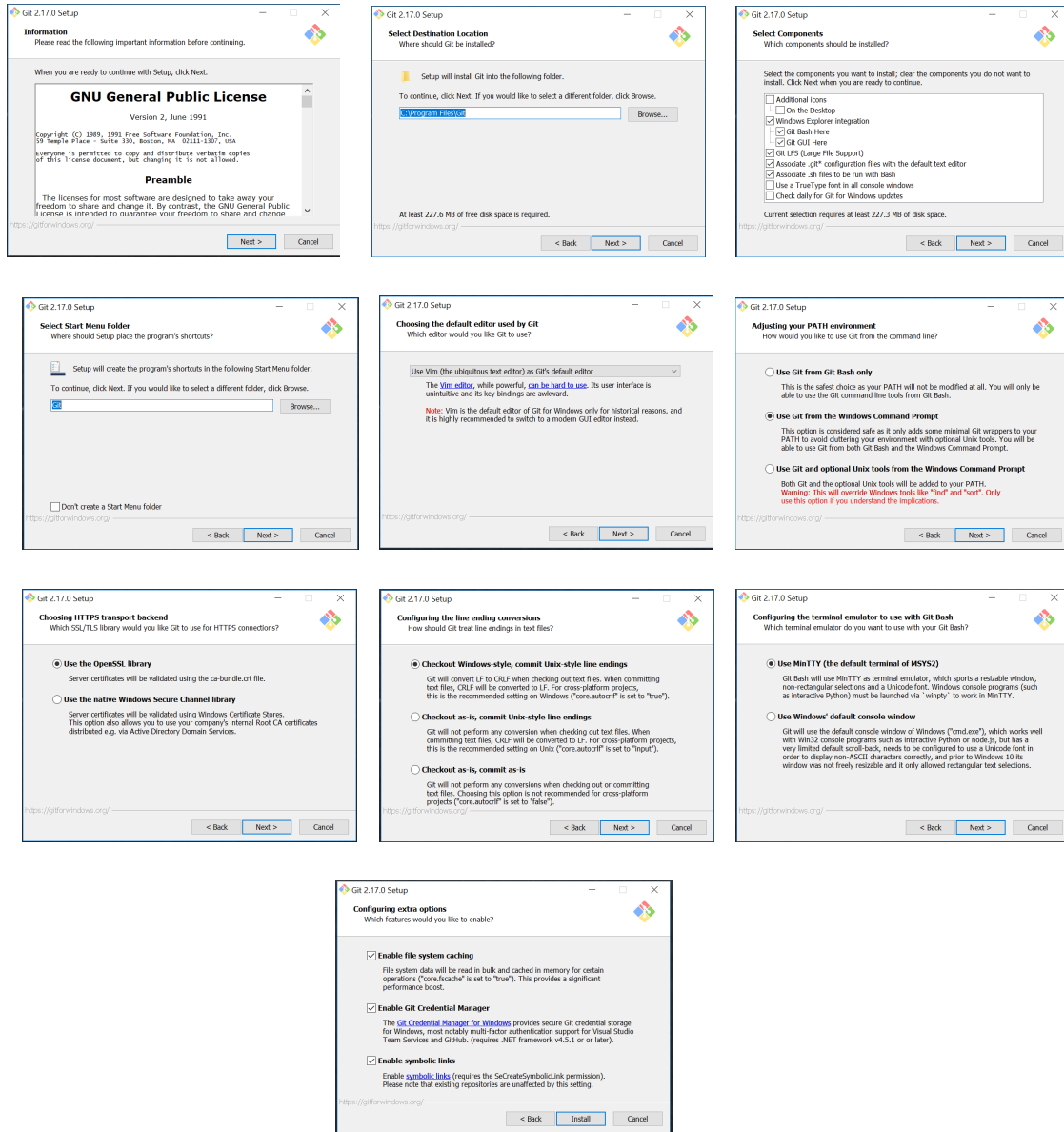
If the new sample works well, you may merge the branch with the master repo. If not, you can just delete the branch and the master repo will still be intact.

A Big Picture



B Windows installation screenshots

Figure 1: Windows installation options



C 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](#)