Introduction to GitHub

Python-do-ECARES
Fabrizio Leone

Introduction

Git and GitHub

- Git is a version control system, i.e. a way to keep track of the whole history of things you do on a file. It is useful to save, manage and edit all the different versions of your project.

- GitHub is a web service that allows to conveniently work with Git. It allows you to create your own directories, see projects of other people and collaborate with them.

- GitHub offers four different subscription plans. We will work with a free one, which means that *everybody* can see what we do. However, with an academic account, you can also create private repositories.

GitHub: What It Does And Does Not Do

- No matter which subscription plan you choose, GitHub offers very limited storage space (you cannot upload files $>$ 100MB). Therefore, it is **not** suitable for storing large files (e.g. datasets). **GitHub is not a substitute for a cloud**.
- GitHub is a platform where to upload mostly **source files** (e.g. .tex, .txt, .m, .R, .do, .py, .doc,...) and light pdf.
- You can read more about Git and GitHub here and here.

GitHub Desktop

- In this course, we interact with GitHub mostly through the GitHub Desktop application.
- GitHub Desktop provides a simple yet powerful desktop interface to GitHub.
- You can download GitHub desktop here.
- You can also interact with GitHub using the Terminal (not covered in this class).

Realistic Workflow

Suppose you are starting out your new project. If you use GitHub, you can

1. Create a local folder with your favourite sub-folders (e.g. code, slides, paper, literature, data,...).
2. Publish some sub-folders on GitHub (e.g. code, slides, paper).
3. Work on your code/paper/slides locally and then save different versions of them on GitHub. (No more: paper_v1, paper_v1.A, paper_v2.89.x7%,...).
4. Scroll through different versions of your files when you need. Collaborate with other people. (No more: paper_v1_myedition, paper_v1_youredition,...).

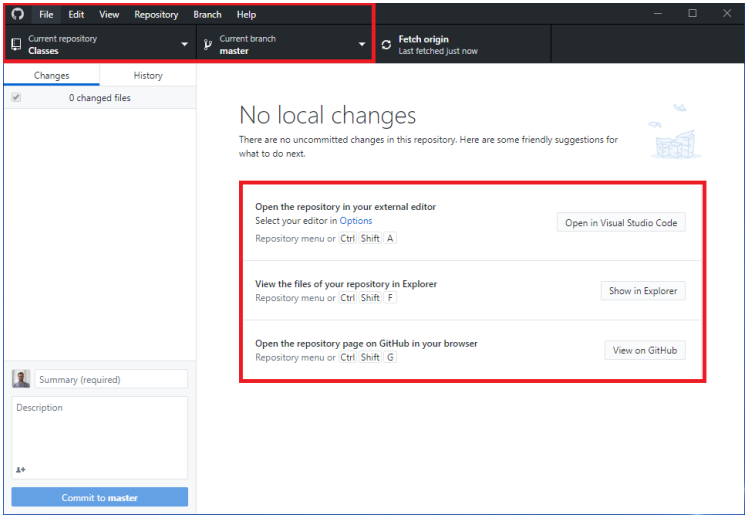
First steps with GitHub Desktop

First Steps With GitHub Desktop

- You should already have (1) opened a GitHub account and (2) downloaded GitHub Desktop.
- If not, please do it now.

Desktop Interface

GitHub Desktop looks like this



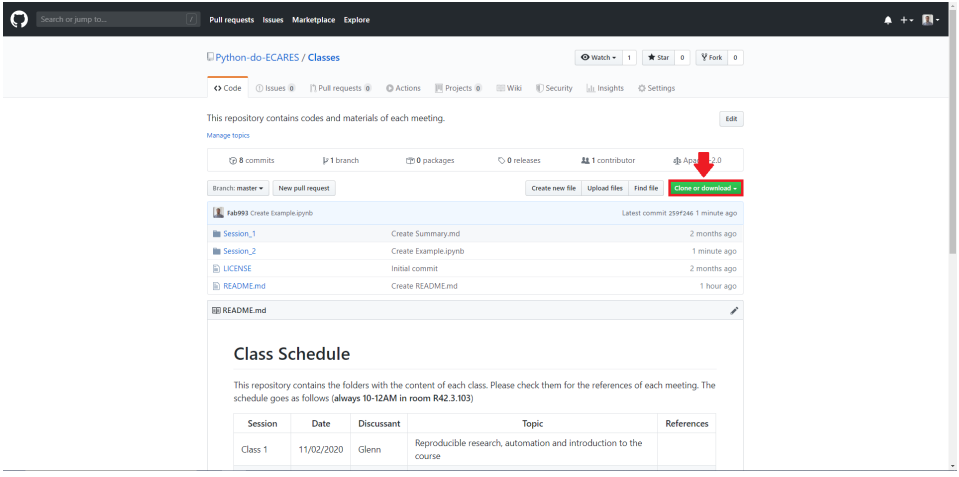
GitHub Desktop For Our Organization

First Steps With GitHub Desktop

- This section shows how to clone a repository, create **your own branch** and how to commit and push changes to it.
- Make sure to familiarise with these steps. You will do this many times in the future.

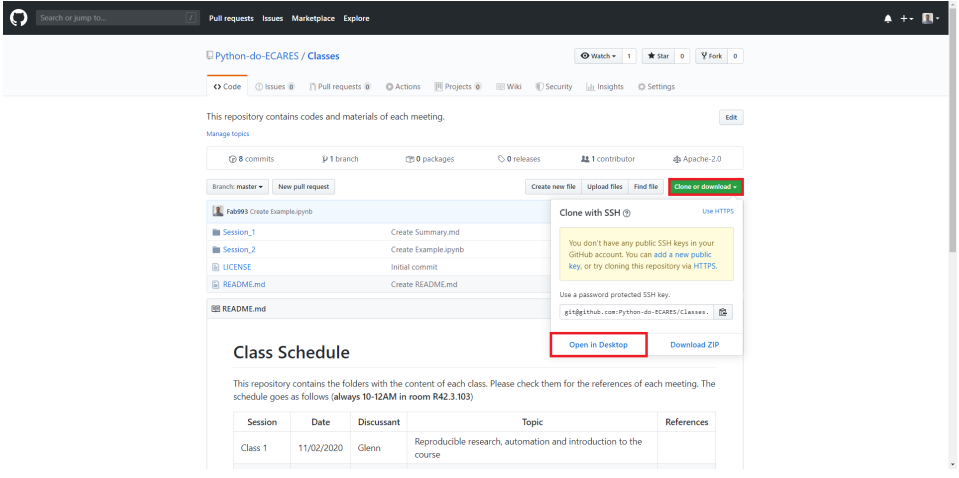
Step 1. Clone Repository

Go to the Classes repository on GitHub and click on the "Clone or download" button.



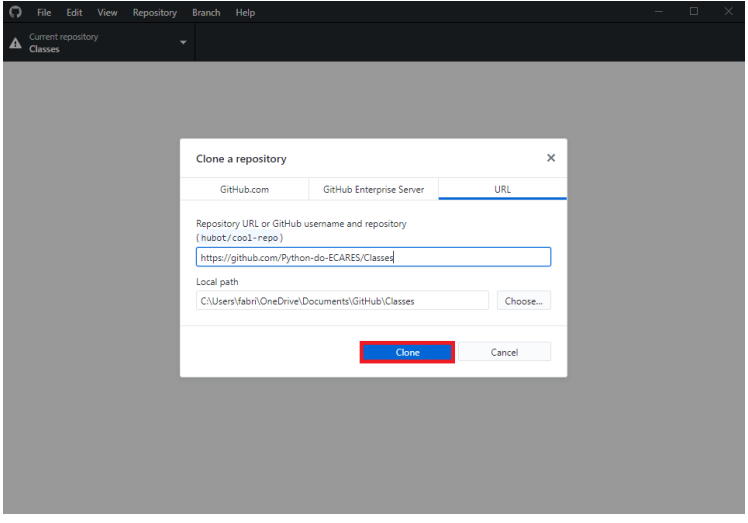
Step 1.A. Open Repository

Choose "Open in Desktop".



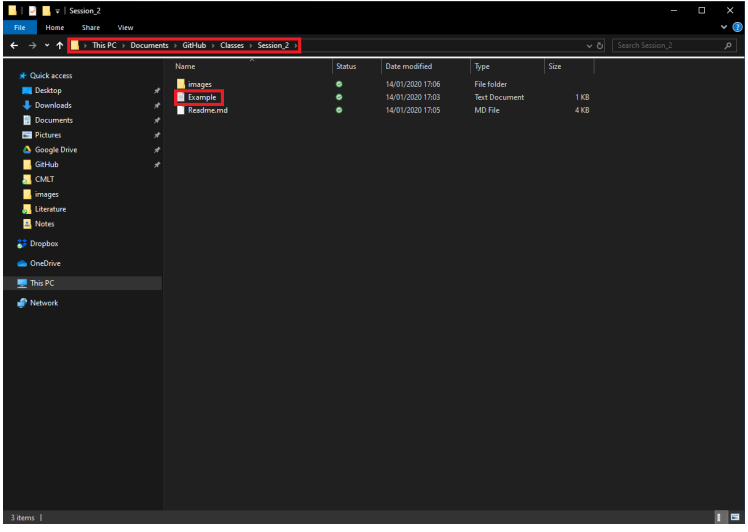
Step 1.B. Clone Repository

Check the local path where to clone the repository and hit ”Clone”.



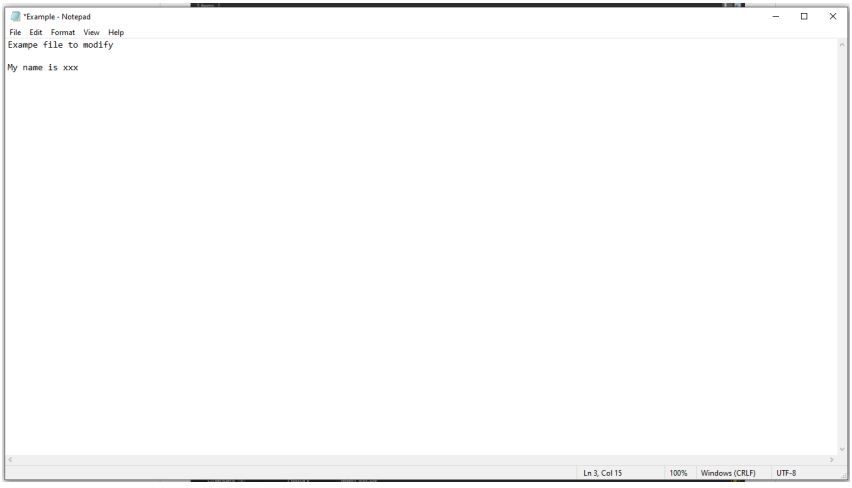
Step 1.C. Open Local Directory

Under the chosen directory, you will see the following files.



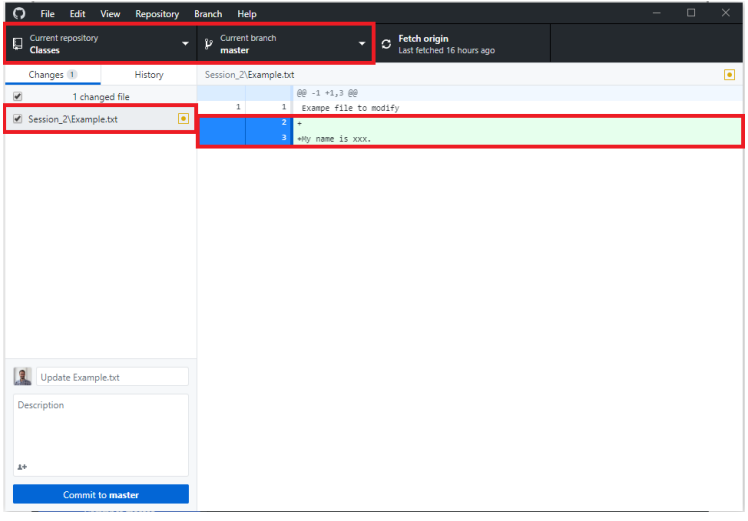
Step 2. Make Changes

Open *Example.txt*. Add a comment line with your name.



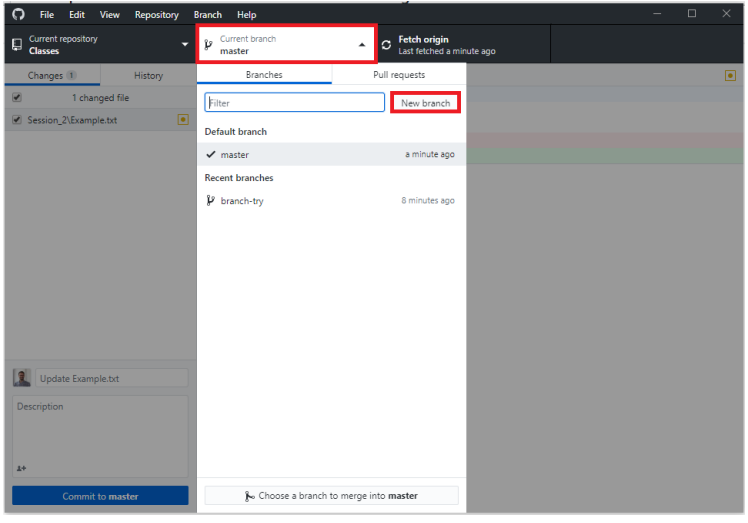
Step 2.A. Save Changes

Save the file. Changes will be displayed in GitHub Desktop as follows.



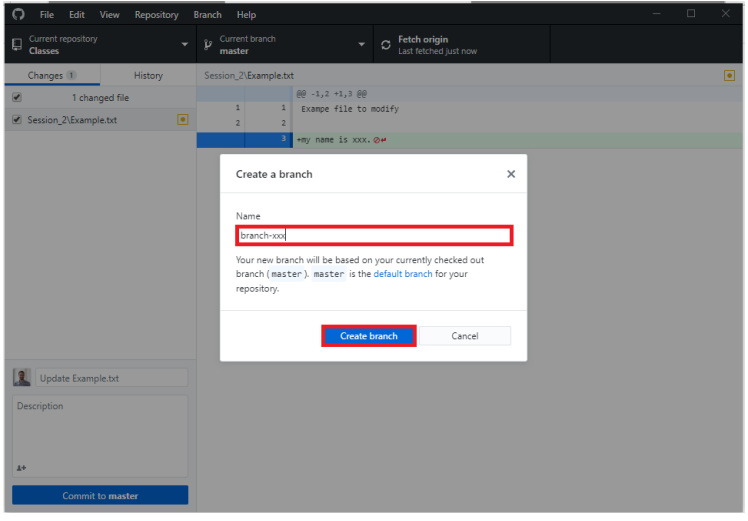
Step 2.B. Create Your Own Branch And Commit Changes

Select ”Master” and hit ”New Branch”.



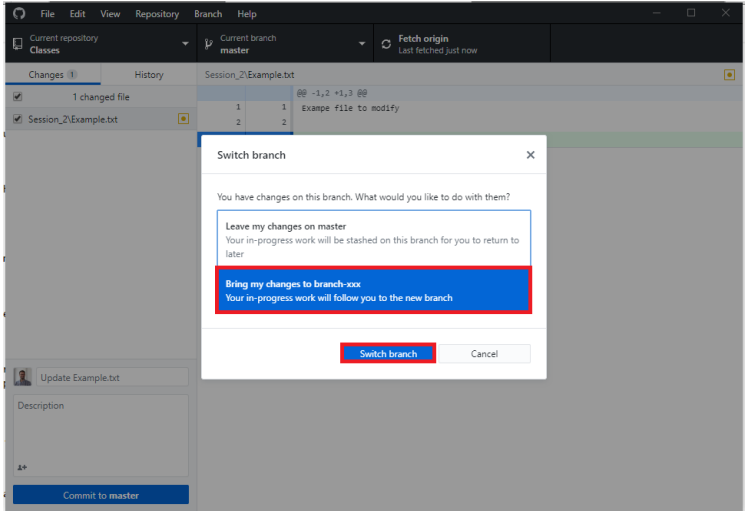
Step 2.C. Name Branch

Give the new branch your name. Then click ”Create Branch”.



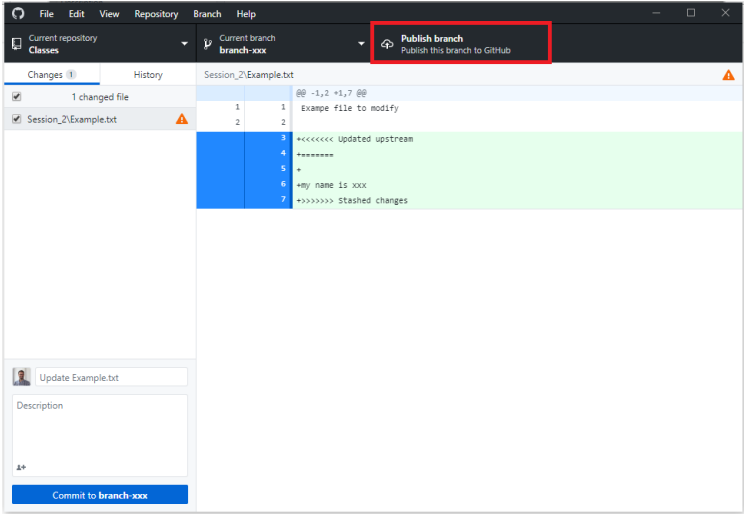
Step 2.D. Switch Branch

Switch changes to your own branch.



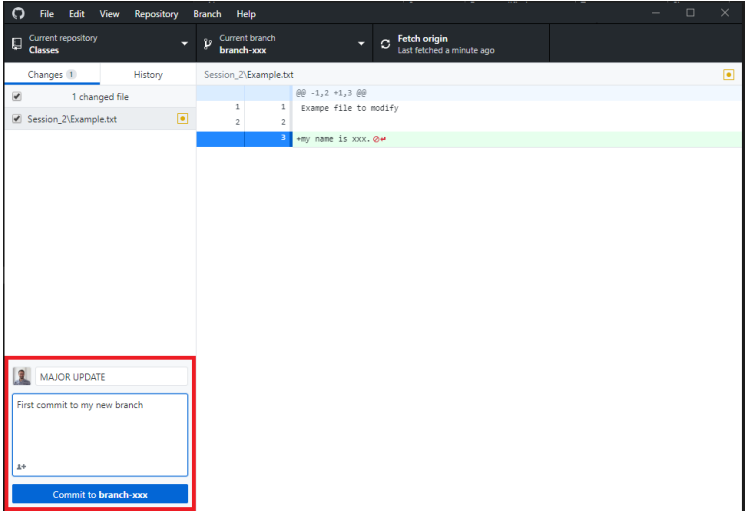
Step 2.E. Publish Branch

Publish your branch online.



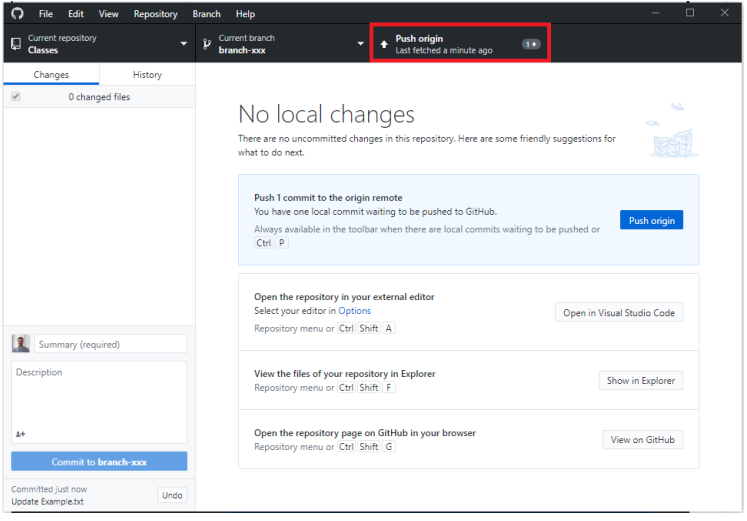
Step 2.F. Commit to Branch

Give Description and summary. Then commit.



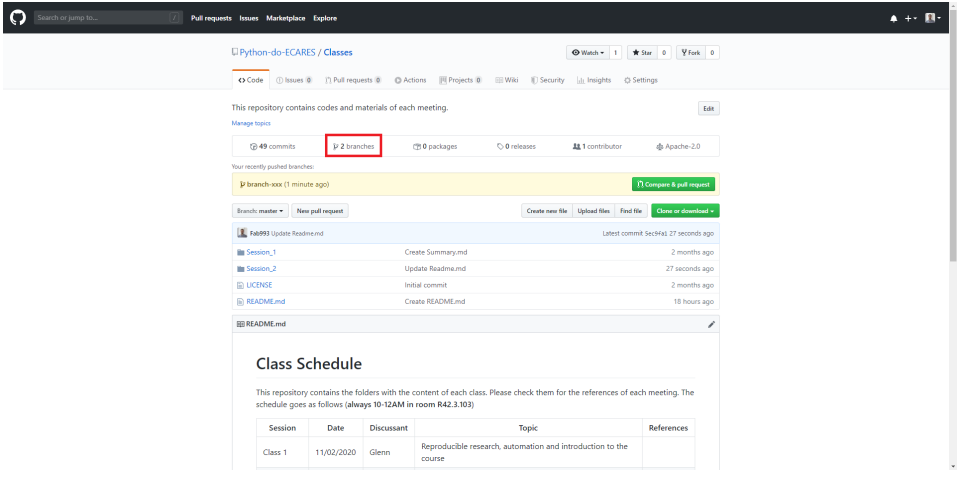
Step 2.G. Push to Branch

Push changes to your branch. You are not allowed to push to the master. If you try, you will get an error.



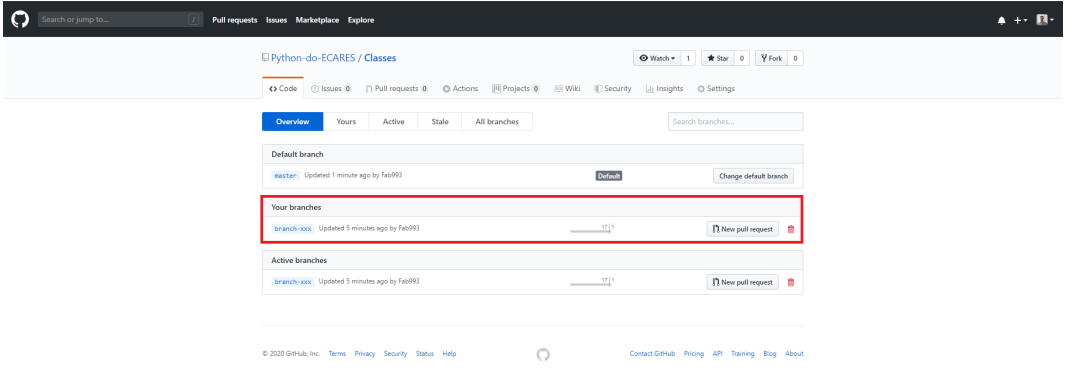
Step 3. Next Steps

Go to "Classes" page. Notice that now there are 2 branches. Click on "branches".



Step 3.A. Done!

Your directory will now appear below "branches" as follows.



Taking Stocks

- You are now able to create and maintain **your own repositories**.
- Please **make sure to only push changes to your own branch during this course**. You are encouraged to collaborate and see what other people is up to, but **never** commit changes directly to other people's branches.
- Good news is that you can always revert changes back if you do so by mistake. This is why GitHub is so useful!

Commit and Push

Commit and Push

- **Committing** and **pushing** are the main two words you have to familiarize with.
- *Committing changes* to a branch, means that you are creating a new version of your file.
- *Pushing changes* means, instead, that you are publishing them online on GitHub. Think of the pushing action as a way of creating different stable releases of your code.
- Only push changes online if you have made a stable change.

Commit and Local Saving

- Notice that **committing to GitHub** and **saving your file locally** are very different things.
- If you save locally, you only change the file on your device. If you commit, you add a "node" to the chain of your versions.
- With this respect, we recommend to commit changes regularly, but also to use **standards**.

Commit Standards

- We encourage you to adopt the following standards to commit tidily.
- "Summary" should be either **Minor Change**, **Major Change** or **Bug Fixes**. The first should indicate small changes in syntaxis or general improvements. The second to major modifications (e.g. add new section or function), while the third is to notify that you have fixed some bug.
- **Description** should briefly explain what the summary refers to.

Commit Standards

- Suppose you **create a new function for data cleaning in your code**. When pushing this change to GitHub, you want to give **Major Change** as summary and ”added function for data cleaning” as description.

- A tidy commit activities will create a full history of changes in GitHub that you can scroll through to check different versions of your code.

- Finally, it will also help other people to understand your work.

Browse Through History

Browse Through History

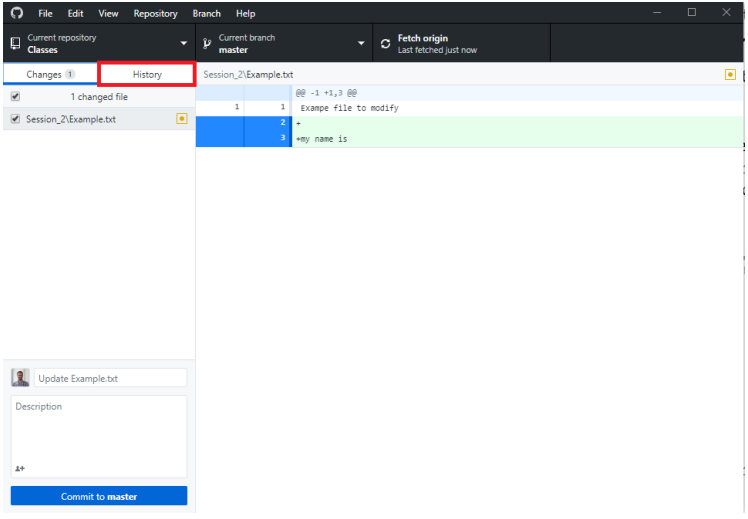
With GitHub Desktop, you can also browse through the entire history of your commits. This is very helpful to

- **Revert back changes**. Imagine you update a code but, at some point, you realise that one of the previous releases was better.
- **Compare different versions**. From time to time, you may want to look back at previous versions of your work (slide, paper, code,...).

A tidy committing activity will help you to easily browse through meaningful versions of your work. If you commit to often, you have to search a lot (versions only differ marginally from one another). If you commit too infrequently, you may lose information.

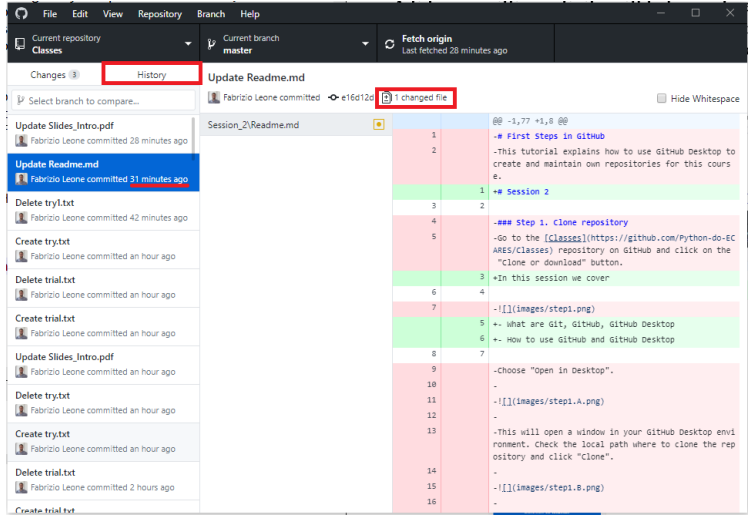
Step1. Revert Back Changes

Open GitHub Desktop at the current repository. Then hit ”History”.



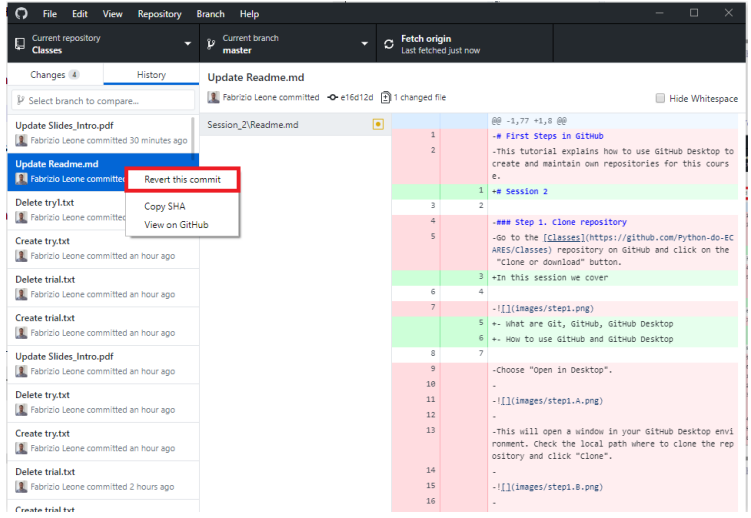
Step2. Revert Back Changes

You can scroll through the whole history of changes you have made up to that moment.



Step3. Revert Back Changes

Click on ”Revert this commit” to go back to that version.



More Details

- You can switch back and forth your versions as many times as you want. **Your local directories will change accordingly**.

- If instead of ”Revert this commit” you hit ”View on GitHub”, you can see all the differences between two versions of your file online.

More Functionalities

More Functionalities

- Communicate with other people (your coauthors, other developers, etc.) by creating **issues**. Look at here for how to do it.

- You find a nice tutorial on how to do both here.